

UNISYS

**A Series
DMSII**

**Application Program
Interfaces
Programming
Guide**

Release 3.8.0
Priced item

October 1989
Distribution Code SE
Printed in U.S. America
5044225.380

UNISYS

A Series

DMSII

**Application Program
Interfaces
Programming
Guide**

Copyright © 1989

Unisys is a trademark of Unisys Corporation

Release 3.8.0

Priced item

October 1989

Distribution Code SE

Printed in U.S. America

5044225.380

Page Status

Page	Issue
iii	Original
iv	Blank
v through xvii	Original
xviii	Blank
xix	Original
xx	Blank
xxi	Original
xxii	Blank
1-1 through 1-7	Original
1-8	Blank
2-1 through 2-15	Original
2-16	Blank
3-1 through 3-9	Original
3-10	Blank
4-1 through 4-5	Original
4-6	Blank
5-1 through 5-5	Original
5-6	Blank
6-1 through 6-27	Original
6-28	Blank
7-1 through 7-6	Original
8-1 through 8-17	Original
8-18	Blank
9-1 through 9-5	Original
9-6	Blank
A-1 through A-17	Original
A-18	Blank
B-1 through B-31	Original
B-32	Blank
C-1 through C-62	Original
D-1 through D-8	Original
Glossary-1 through 20	Original
Bibliography-1 through 3	Original
Bibliography-4	Blank
Index-1 through 12	Original

About This Guide

Purpose

The *A Series DMSII Application Program Interfaces Programming Guide* explains how to write effective and efficient application programs that access and manipulate a Data Management System II (DMSII) database using either the DMSII interpretive interface or the DMSII language extensions.

Scope

This guide presents programming considerations, efficiency guidelines, and status messages for both the DMSII interpretive interface and the DMSII language extensions. Details of the DMSII interpretive interface, including generating the DMINTERPRETER library, using entry points, and coding programs can be found in the *A Series DMSII Interpretive Interface Programming Reference Manual*. Specific information about coding the DMSII language extensions can be found in Volume 2 of the individual programming language reference manuals.

Some of the material covered in this document was previously published in the *A Series DMSII User Language Interface Programming Guide*.

Audience

This guide is written for application programmers who are responsible for programs that report or update information in a DMSII database. The guide can also prove useful to database administrators.

Prerequisites

Before programming DMSII applications, you should be familiar with basic A Series concepts, basic DMSII concepts, the Data and Structure Definition Language (DASDL), and the programming language you intend to use. You should also know how to use the A Series Command and Edit (CANDE) message control system (MCS) to enter code, and know how to use CANDE and the Work Flow Language (WFL) to compile and run jobs.

How to Use This Guide

This guide has been designed to help you code effective and efficient application programs using either the DMSII interpretive interface or the DMSII language extensions.

If you are new to database or DMSII applications programming, read Sections 1, 2, and 3 before you begin designing your program.

- Use the information in Section 1 to select an application programming interface.
- Review Section 2 for DMSII programming terms and concepts. These terms and concepts help you understand the rationale behind the task-specific programming guidelines.
- Read Section 3 for an overview of the data management tasks available through the DMSII application program interfaces.

If you are an experienced DMSII programmer, read Section 3 to review the data management tasks. Sections 4 through 8 contain task-specific guidelines. Use either the table of contents or the index to locate the specific task. Unless noted, the programming guidelines apply to both interfaces.

While Section 9 presents guidelines for handling exception and error conditions within your application program, Appendix A details the actual exception and error messages returned by the Accessroutines to your application program.

Program fragments are used for text examples. Complete programs, using both the interpretive interface and language extensions, are listed in the appendixes. Appendix B contains ALGOL, COBOL74, and FORTRAN77 programs that use the interpretive interface. Appendix C contains ALGOL, COBOL74, and RPG programs that use language extensions. Consult the COBOL and COBOL85 language reference manuals for examples of COBOL and COBOL85 programs that use language extensions.

The separate COBOL languages (COBOL, COBOL74, and COBOL85) are referred to collectively in the text, index, and glossary as COBOL. However, if there are any differences, the specific ANSI version of the language is identified.

For ease of reading, direct references to interpretive interface entry points or language extension statements are kept to a minimum. Instead, the text refers to the function, such as a find operation. When it is necessary to refer to an entry point, the entry point is identified by its COBOL name in the DATABASE/DMINTERPRETER file.

The entry points for transferring data from the user work area to user-declared variables are referred to collectively as DBGET entry points. The entry points for transferring data from user-declared variables to the user work area are referred to collectively as DBPUT entry points.

Variations of the find operation are used to show the features of all the search operations (delete, find, lock, modify, and secure).

Metatokens are used for describing syntactical elements. For example, the metatoken < data set qualified name > refers to the syntax of a data set name. Syntax diagrams

and syntax elements are noted in the index under the appropriate metatoken. An explanation of railroad diagrams (a Unisys method of depicting syntax) is found in Appendix D.

In the text, italics are used to introduce new terms or to identify a word used in an unusual manner.

This guide uses the word *access* as both a noun and a verb. As a noun, it always appears with a capital A and refers to a database part. As a verb, it means to use or retrieve something.

This guide also uses the word *recreate* as both a verb and a DMSII operation. As a verb, it always appears with a hyphen (re-create). As a DMSII operation it appears without the hyphen (recreate operation).

Throughout the document, *A Series* is used to refer collectively to A Series and B 7900 systems. Unless otherwise noted, all references to related documentation within the text are for A Series product information. The glossary spells out and defines all acronyms used in the guide.

For a broader view of programming DMSII applications, you can also do the following:

- Consult the *A Series DMSII Technical Overview* for background information and information about designing and maintaining databases. The DMSII Technical Overview also contains several examples of COBOL74 application programs that use language extensions.
- Read the *A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* for DASDL option specifics.
- Refer to the *A Series DMSII Utilities Operations Guide* for detailed information on the use of DMSII programs.
- Use the information in the individual programming language reference manuals or in the *A Series DMSII Interpretive Interface Programming Reference Manual* to code your application programs.

Organization

This guide is organized into nine sections. Each section explores topics and tasks that pertain both to the DMSII interpretive interface and to the DMSII language extensions. In addition, a glossary, a bibliography, four appendixes, and an index appear at the end of the guide.

Section 1. Comparing the DMSII Application Program Interfaces

Section 1 explores the similarities and differences between the DMSII interpretive interface and the DMSII language extensions. It identifies the basic modules and advantages of each application program interface.

Section 2. Understanding Basic DMSII Programming Terms and Concepts

Section 2 explains fundamental DMSII programming terms and concepts, such as current path and database.

Section 3. Identifying Data Management Tasks

Section 3 provides an overview of the data management tasks available through the DMSII application program interfaces, including a discussion about when each task should be used. Each task is detailed in later sections.

Section 4. Opening and Closing a Database

Section 4 presents recommendations for invoking, opening, and closing a database.

Section 5. Setting the Current Path

Section 5 details how to use the set operation to avoid unnecessary NOTFOUND exceptions and how to use find operations to reset invalid current paths. The section also briefly describes the various operations that change the current path.

Section 6. Retrieving Records through Search Operations

Section 6 explores the various techniques used to search for and retrieve records and data values.

Section 7. Updating the Database

Section 7 presents guidelines for adding, changing, and deleting data.

Section 8. Recovering an Audited Database

Section 8 details how to use the restart data set, audit trail, and transaction processing operations for effective and efficient recovery of audited databases.

Section 9. Handling Exception and Error Conditions

Section 9 contains questions to use to determine whether DMSII or your application program should handle an exception or error condition.

Appendix A. DMSII Exceptions and Errors

This appendix lists the exceptions and errors returned to an application program by the Access routines.

Appendix B. Sample Programs Using the Interpretive Interface

This appendix contains the required DASDL source file and the listing for ALGOL, COBOL74, and FORTRAN77 application programs that demonstrate possible coding techniques for programs that use the interpretive interface.

Appendix C. Sample Programs Using Language Extensions

This appendix contains the required DASDL source file and the listing for ALGOL, COBOL74, and RPG application programs that demonstrate possible coding techniques for programs that use the language extensions.

Appendix D. Understanding Railroad Diagrams

Appendix D explains how to read railroad diagrams.

Related Product Information

The following list contains companion Unisys documents for this guide:

A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation (form 1169844)

This manual describes the basic features of the Extended ALGOL programming language. This manual is written for programmers who are familiar with programming concepts.

A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces (form 1170099)

This manual describes extensions to the Extended ALGOL language designed to allow application programs to interface with the Advanced Data Dictionary System (ADDS), the Communications Management System (COMS), the Data Management System II (DMSII), the Screen Design Facility Plus (SDF Plus), or the Semantic Information Manager (SIM). This manual is written for programmers who are familiar with Extended ALGOL programming language concepts and terms.

A Series CANDE Operations Reference Manual (form 1169869)

This manual describes how CANDE operates to allow generalized file preparation and updating in an interactive, terminal-oriented environment. This manual is written for a wide range of computer users who work with text and program files.

A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation (form 1169877)

This manual describes the basic features of the standard COBOL ANSI-74 programming language, which is fully compatible with the American National Standard, X3.23-1974. This manual is written for programmers who are familiar with programming concepts.

A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces (form 1180619)

This manual describes the extensions to the standard COBOL74 language. These extensions are designed to allow application programs to interface with the Advanced Data Dictionary System (ADDS), the Communications Management System (COMS), the Data Management System II (DMSII), the DMSII Transaction Processing System (TPS), the Screen Design Facility (SDF), the Screen Design Facility Plus (SDF Plus), and the Semantic Information Manager (SIM) products. This manual is written for

programmers who are familiar with COBOL74 programming language concepts and terms.

A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation (form 5031503)

This manual describes the features of the COBOL ANSI-85 programming language, as implemented on Unisys A Series systems. Volume 1 describes the basic implementation of the language. This manual is written for programmers who are familiar with programming concepts.

A Series COBOL ANSI-85 Programming Reference Manual, Volume 2: Product Interfaces (form 5044308)

This manual describes the extensions to the standard COBOL85 language. These extensions are designed to allow application programs to interface with the Communications Management System (COMS) and the Data Management System II (DMSII) products. This manual is written for programmers who are familiar with COBOL85 programming language concepts and terms.

A Series COBOL Programming Reference Manual (form 1169786)

This manual provides a complete description of COBOL as developed by the CODASYL Committee and described by the American National Standards, X3.23-1968. This manual is written for programmers who are familiar with programming concepts.

A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual (form 1163805)

This manual provides instructions for defining and maintaining a Data Management System II (DMSII) database through DASDL. This manual is written for database administrators and staff.

A Series DMSII Interpretive Interface Programming Reference Manual (form 1163813)

This manual details the DMSII interpretive interface functions, including generating the DMINTERPRETER library, using entry points, and coding programs. This manual is written for application programmers and database administrators who are already knowledgeable in the basic concepts of DMSII.

A Series DMSII Technical Overview (form 5044191)

This overview describes the Data Management System II (DMSII) environment, which includes DMSII databases and software. This overview is written for end users, system programmers, database designers, and database administrators who need information on creating, changing, restoring, accessing, and maintaining a DMSII database.

A Series DMSII Transaction Processing System (TPS) Programming Guide (form 1164043)

This guide describes the various modules of the TPS and provides information on the TPS library of transaction processing procedures. This manual is written for experienced systems programmers who are familiar with Data Management System II (DMSII).

A Series DMSII Utilities Operations Guide (form 1163839)

This guide describes how to maintain relationships between the various data elements in the database. This guide is written for database administrators or programmers who are responsible for database integrity and recovery.

A Series Extended Retrieval with Graphic Output (ERGO) Operations Guide (form 1164027)

This guide provides information on the daily use of ERGO to access Data Management System II (DMSII) databases. This guide is written for managers, programmers, analysts, and support personnel involved in preparing reports from DMSII databases.

A Series FORTRAN77 Programming Reference Manual (form 1222666)

This manual describes the FORTRAN77 programming language, which is fully compatible with the American National Standard, X3.9-1978. This manual is written for programmers who are familiar with programming concepts.

A Series Report Program Generator (RPG) Programming Reference Manual, Volume 1: Basic Implementation (form 1169760)

This manual describes the basic features of the RPG programming language, including both RPG I and RPG II dialects. This manual is written for applications programmers who are familiar with programming concepts.

A Series Report Program Generator (RPG) Programming Reference Manual, Volume 2: Product Interfaces (form 1170115)

This manual describes extensions to the RPG language designed to allow application programs to interface with the Data Management System II (DMSII) or the Communications Management System (COMS). This manual is written for applications programmers who are familiar with the RPG programming language concepts and terms.

A Series Security Administration Guide (form 1195195)

This guide describes systems-level security features and gives suggestions for using those features. It provides administrators with the information necessary to set and implement effective security policy. This guide is written for system administrators, security administrators, and those responsible for establishing and implementing security policy.

A Series Security Features Operations and Programming Guide (form 1195203)

This guide describes the security features available to users and provides instructions on their use. This guide is written for users who are responsible for maintaining the security of their individual programs and data.

A Series Work Flow Administration and Programming Guide (form 1170149)

This guide discusses the facilities for task initiation and control, including an overview of tasking, a discussion of process management, information on controlling specific processes, and a detailed discussion of task attributes. This guide is written for system administrators, operators, and programmers.

A Series Work Flow Language (WFL) Programming Reference Manual (form 1169802)

This manual presents the complete syntax and semantics of the WFL. This language is used to construct jobs that compile or run programs written in other languages, and is used to construct jobs that perform library maintenance such as copying files. This manual is written for individuals who have some experience with programming in a block-structured language such as ALGOL and who know how to create and edit files using CANDE or the Editor.

Contents

About This Guide	v
 Section 1. Comparing the DMSII Application Program Interfaces	
Understanding the Basics of the Language Extensions	1-2
Using SYSTEM/DMINTERFACE at Compile Time	1-4
Using the Accessroutines at Run Time	1-4
Understanding the Basics of the Interpretive Interface	1-4
Using the Standard Language Compiler at Compile Time	1-7
Using the Accessroutines at Run Time	1-7
Tasks Performed by the Accessroutines	1-7
 Section 2. Understanding Basic DMSII Programming Terms and Concepts	
Separating Data Definition from Data Manipulation	2-1
Using DMSII to Separate Data Definition from Data Manipulation	2-1
Defining the Database Schema with DASDL	2-2
Creating the Database Schema	2-2
Using the Description File	2-3
Changing the Database Schema	2-3
Monitoring Database Operations	2-3
Manipulating Data Values in Application Programs ...	2-4
Invoking and Opening the Database	2-4
Investigating the Database Schema	2-5
Distinguishing between Audited and Unaudited Databases	2-5
Processing Transactions	2-5
Processing Transactions in Audited Databases ..	2-5
Transaction Processing Guidelines for Audited Databases	2-6
Processing Transactions in Unaudited Databases ..	2-6
Recovering a Database	2-7
Recovering Audited Databases	2-7
Recovering Unaudited Databases	2-7
Understanding Current Path and Current Record	2-8
Determining What Structures Are Available to a Program	2-8
Relating the Current Path to the Current Record	2-8
Identifying the Beginning and End of a Structure	2-9
Maintaining Separate Paths for Each Structure	2-9
Defining the Status of the Current Path	2-12
Defining the Status of the Current Record	2-13
Locking and Securing a Record	2-13

Freeing a Record	2-14
If a Record Is Locked Outside Transaction State	2-14
If a Record Is Locked Within Transaction State	2-14
Locking, Securing, and Freeing a Structure	2-15
Creating and Re-creating a Record in an Application Program ..	2-15
 Section 3. Identifying Data Management Tasks	
Dividing Tasks by Interface and Type of Database	3-2
Identifying Interpretive Interface Data Management Operations ..	3-3
Standard Entry Point Operations	3-4
Data Transfer Entry Point Operations	3-5
Exception Handling Entry Point Operations	3-7
Using Language Extensions as Data Management Statements and Commands	3-7
 Section 4. Opening and Closing a Database	
Invoking a Database	4-1
Invoking either a Physical or a Logical Database	4-1
Overriding the Database Named in a Database Invocation	4-2
Sharing the Same Database Declaration	4-4
Opening a Database	4-4
Closing a Database	4-5
 Section 5. Setting the Current Path	
Using a Set Operation	5-2
Using Search Operations to Set the Current Path for a Data Set ..	5-2
Using Search Operations to Set the Current Path for a Set or Subset	5-3
Adjusting the Path for Deleted Records	5-5
 Section 6. Retrieving Records through Search Operations	
Coding a Search Operation	6-1
Understanding What Happens to a Retrieved Record	6-2
Identifying the Types of Searches	6-2
Searching Serially by Physical Location	6-3
Examining Each Record in a Serial Search	6-3
Using a Forward Serial Search	6-3
Using a Backward Serial Search	6-4
Using the Serial Parameters	6-4
Retrieving the First Physical Record	6-5
Retrieving the Next Physical Record	6-5
Retrieving the Prior Physical Record	6-6
Retrieving the Last Physical Record	6-6

Searching Randomly by Selection Expression	6-7
Defining the Components of Selection Expressions	6-7
Selecting a Target Structure	6-8
Selecting Key Items	6-8
Using Complex Key Items	6-9
Using Key Item Order	6-9
Using Duplicate Key Items	6-9
Using the Same Key Item More Than Once	6-10
Using Nonkey Items in Selection Expressions ...	6-10
Testing for Relationships	6-10
Selecting a Target Value	6-11
Finding Exact Matches	6-11
Matching Alphanumeric Items Exactly	6-12
Matching Numeric Items Exactly	6-12
Finding an Approximate Match	6-12
Using Boolean Operators	6-13
Using the NOT Operator	6-13
Using the AND Operator to Test for More than One Criterion	6-14
Using the OR Operator to Test for More than One Criterion	6-15
Using AND, OR, and NOT Operators Together ...	6-16
Combining Serial and Random Searches	6-17
Find Key Of Operations for Sets and Subsets	6-18
Using Binary Search Techniques for Sets and Subsets	6-18
Using Partial Binary Searches for Duplicate Keys	6-20
Optimizing a Search with Reblocking and Readheads	6-22
Examples of Locating Data Set Records	6-22
Understanding Search Operation Differences for the Interpretive Interface	6-24
Evaluating Selection Expressions	6-24
Finding a Data Set Using Selection Expressions	6-25
Terminating Serial Searches Early	6-25
Performing a Binary Search on a Complex Key	6-25
Using an Automatic Find Key Of Operation	6-26
Guidelines for Search and Retrieval Operations	6-27

Section 7. Updating the Database

Selecting the Appropriate Means of Retrieving the Record	7-1
When the Database Is Open for Inquiry Only	7-1
When the Database Is Open for Updates	7-2
Preventing Deadlocks	7-2
Adding New Records to a Data Set	7-2
Changing the Value of Items	7-3
Changing a Nonrequired Item	7-3
Changing a Required Item	7-4
Deleting Data Records	7-5
Adding and Deleting Key Entries in Manual Subsets	7-5

Section 8. Recovering an Audited Database

Understanding the ABORT Task	8-1
Conventional Aborts	8-2
Aborts Where BACKOUTTOSPT Is Set	8-2
Understanding a Halt/Load	8-2
Defining Recovery Options through DASDL	8-3
Setting the DASDL AUDIT Option for Audited Databases	8-4
Setting the DASDL INDEPENDENTTRANS Option for Concurrency Control	8-5
Setting the DASDL REAPPLYCOMPLETED Option for Recovery	8-5
Understanding the Purpose of a Restart Data Set	8-5
Identifying the Last Successful Transaction	8-6
Identifying Your Program	8-6
Example of the DASDL Source Code for a Restart Data Set	8-6
Adding and Deleting Records in the Restart Data Set	8-7
Knowing When to Read the Restart Data Set Records	8-7
Identifying the Restart Procedure Tasks	8-8
Using Transaction State	8-8
Beginning and Ending Transaction State	8-9
Using the AUDIT Option to Write Records to the Restart Data Set	8-10
Forcing a Syncpoint	8-10
Multitasking in Transaction State	8-11
Understanding Language Extension Recovery Commands	8-11
Example of a Restartable Program with an Audited Database ...	8-12

Section 9. Handling Exception and Error Conditions

Distinguishing between Exceptions and Errors	9-1
Fatal Conditions	9-1
Nonfatal Conditions	9-1
Understanding SYSTEMERROR Causes and Responses	9-2
Understanding INTEGRITYERROR Causes and Responses	9-2
Deciding Whether to Mask Messages	9-3
Handling Exception or Error Conditions through DMSII	9-3
Handling Exception or Error Conditions through Your Program ..	9-4
Examining the DMSTATUS Register	9-4
Examining the DBSTATUS Exception Word	9-4

Appendix A. DMSII Exceptions and Errors

Categorizing Exceptions and Errors	A-2
ABORT Subcategories	A-2
AUDITERROR Subcategories	A-3
CLOSEERROR Subcategories	A-4
DATAERROR Subcategories	A-4
DEADLOCK Subcategories	A-4

DUPLICATES Subcategories	A-5
FATALERROR Subcategories	A-5
INTEGRITYERROR Subcategories	A-5
INTLIBERROR Subcategories	A-6
INUSE Subcategories	A-7
IOERROR Subcategories	A-8
KEYCHANGED Subcategories	A-9
LIMITERROR Subcategories	A-9
NORECORD Subcategories	A-10
NOTFOUND Subcategories	A-10
NOTLOCKED Subcategories	A-11
OPENERROR Subcategories	A-11
READONLY Subcategories	A-14
SECURITYERROR Subcategories	A-15
SYSTEMERROR Subcategories	A-15
VERSIONERROR Subcategories	A-17
 Appendix B. Sample Programs Using the Interpretive Interface	
DASDL Database for Interpretive Interface Sample Programs ...	B-2
ALGOL Application Program Using the Interpretive Interface ...	B-5
COBOL74 Application Program Using the Interpretive Interface .	B-15
FORTRAN77 Application Program Using the Interpretive Interface	B-25
 Appendix C. Sample Programs Using Language Extensions	
DASDL Database for the ALGOL Language Extension Program ..	C-2
ALGOL Application Program Using Language Extensions	C-7
COBOL74 Application Program Using Language Extensions	C-16
DASDL Database for the RPG Language Extension Program	C-25
RPG Application Program Using Language Extensions	C-26
 Appendix D. Understanding Railroad Diagrams	
What Are Railroad Diagrams?	D-1
Constants and Variables	D-2
Constraints	D-2
Following the Paths of a Railroad Diagram	D-5
Railroad Diagram Examples with Sample Input	D-6
 Glossary	1
 Bibliography	1
 Index	1

Figures

1-1.	Database Processing with Language Extensions	1-3
1-2.	Database Processing with the Interpretive Interface	1-6
2-1.	Transaction State	2-6
2-2.	Current Path and Current Record	2-8
2-3.	Beginning and End of a Structure	2-9
2-4.	Multiple Paths in One Application Program (One for Each Structure)	2-10
2-5.	Multiple Uses of the Same Structure in One Application Program	2-11
2-6.	Current Paths When Multiple Application Programs Use the Same Structures	2-12
2-7.	Space Between Records	2-13
5-1.	Resetting the Current Path for a Data Set	5-2
5-2.	Resetting the Current Path for a Set	5-4
6-1.	Forward Serial Search	6-3
6-2.	Backward Serial Search	6-4
6-3.	Retrieving the First Physical Record	6-5
6-4.	Retrieving the Next Physical Record	6-6
6-5.	Retrieving the Prior Physical Record	6-6
6-6.	Retrieving the Last Physical Record	6-6
6-7.	Binary Searches	6-19
6-8.	Find Operations That Force a Partial Binary Search	6-21
6-9.	Locating Data Set Records through a Set	6-23
D-1.	Railroad Constraints	D-5

Tables

2-1.	Database Monitoring Programs	2-3
3-1.	Data Management Operations Listed by Task	3-1
3-2.	Application Program Data Management Tasks	3-3
3-3.	Standard Entry Points by Task	3-4
3-4.	Data Transfer Entry Points by Task	3-5
3-5.	Exception Handling Entry Points by Task	3-7
3-6.	DMSII Language Extension Operations	3-8
5-1.	Data Management Operations that Set the Current Path	5-1
6-1.	Search and Retrieval Operations	6-1
6-2.	Serial Search Parameters	6-5
6-3.	Relational Operators in a Selection Expression	6-11
6-4.	Efficient Combinations for Search Operations Using Ascending Keys	6-12
6-5.	Efficient Combinations for Search Operations Using Descending Keys	6-13
6-6.	Truth Table for Boolean Operators	6-13
6-7.	Combined Search and Retrieval Operations	6-17
8-1.	DASDL Options for Audited Databases	8-4
8-2.	Language Extensions Auditing Operations	8-12
A-1.	Major Categories for Exceptions and Errors by Category Number	A-2
A-2.	ABORT Subcategories	A-3
A-3.	AUDITERROR Subcategories	A-3
A-4.	CLOSEERROR Subcategories	A-4
A-5.	DATAERROR Subcategories	A-4
A-6.	DEADLOCK Subcategories	A-5
A-7.	DUPLICATES Subcategories	A-5
A-8.	INTEGRITYERROR Subcategories	A-6
A-9.	INTLIBERROR Subcategories	A-6
A-10.	INUSE Subcategories	A-8
A-11.	IOERROR Subcategories	A-8
A-12.	KEYCHANGED Subcategories	A-9
A-13.	LIMITERROR Subcategories	A-9
A-14.	NORECORD Subcategories	A-10
A-15.	NOTFOUND Subcategories	A-11
A-16.	NOTLOCKED Subcategories	A-11
A-17.	OPENERERROR Subcategories	A-12
A-18.	READONLY Subcategories	A-15
A-19.	SECURITYERROR Subcategories	A-15
A-20.	SYSTEMERROR Subcategories	A-15
A-21.	VERSIONERROR Subcategories	A-17

Section 1

Comparing the DMSII Application Program Interfaces

There are two DMSII application program interfaces: the DMSII interpretive interface and the DMSII language extensions. Each interface is divided into a compile-time interface and a run-time interface. Both the interpretive interface and the language extensions use the Accessroutines as their run-time interface.

For your application program to use the interpretive interface, the standard compiler for the application language must be able to import library objects. For your application program to use language extensions, the standard compiler for the application language must be specifically modified to support the necessary DMSII extensions.

- Application programs coded in ALGOL, C, COBOL, COBOL74, COBOL85, FORTRAN77, Pascal, or PL/1 can use the interpretive interface.
- Application programs coded in ALGOL, COBOL, COBOL74, COBOL85, and RPG can use language extensions.
- If you code your application program in ALGOL, COBOL, COBOL74, or COBOL85, you can use both interfaces in the same program.

The main difference between the two program interfaces is the point at which information describing the database is passed to the program. Programs using the interpretive interface receive the description at run time. Programs using language extensions receive the description at compile time.

Each database has its own control file, description file, and DMSUPPORT library. Each structure within the database has its own data file. For the interpretive interface, each database has its own DMINTERPRETER library.

This section describes the basics of each programming interface and lists the exclusive advantages of each interface.

Understanding the Basics of the Language Extensions

Traditionally, application programs have accessed DMSII databases through DMSII language extensions. The DMSII language extensions are specially modified user language compilers that recognize DMSII constructs and translate them directly into calls on the database.

The following list gives the advantages of using language extensions to access a database:

- DMSII optimizes application program run-time performance.
- Programs written in supported COBOL languages can access the database directly – they do not need to use the user work area. ALGOL programs, using variations of the FIND statement, can also bypass the user work area.
- The database description is known at compile time.
- Programs can use independent transaction operations.
- Data type checking can be directly enforced.
- Programs can use links and embedded manual subsets.

Figure 1-1 shows the modules and timing involved in compiling an application program that uses language extensions. The compile-time interface is SYSTEM/DMINTERFACE. The run-time interface is the Accessroutines.

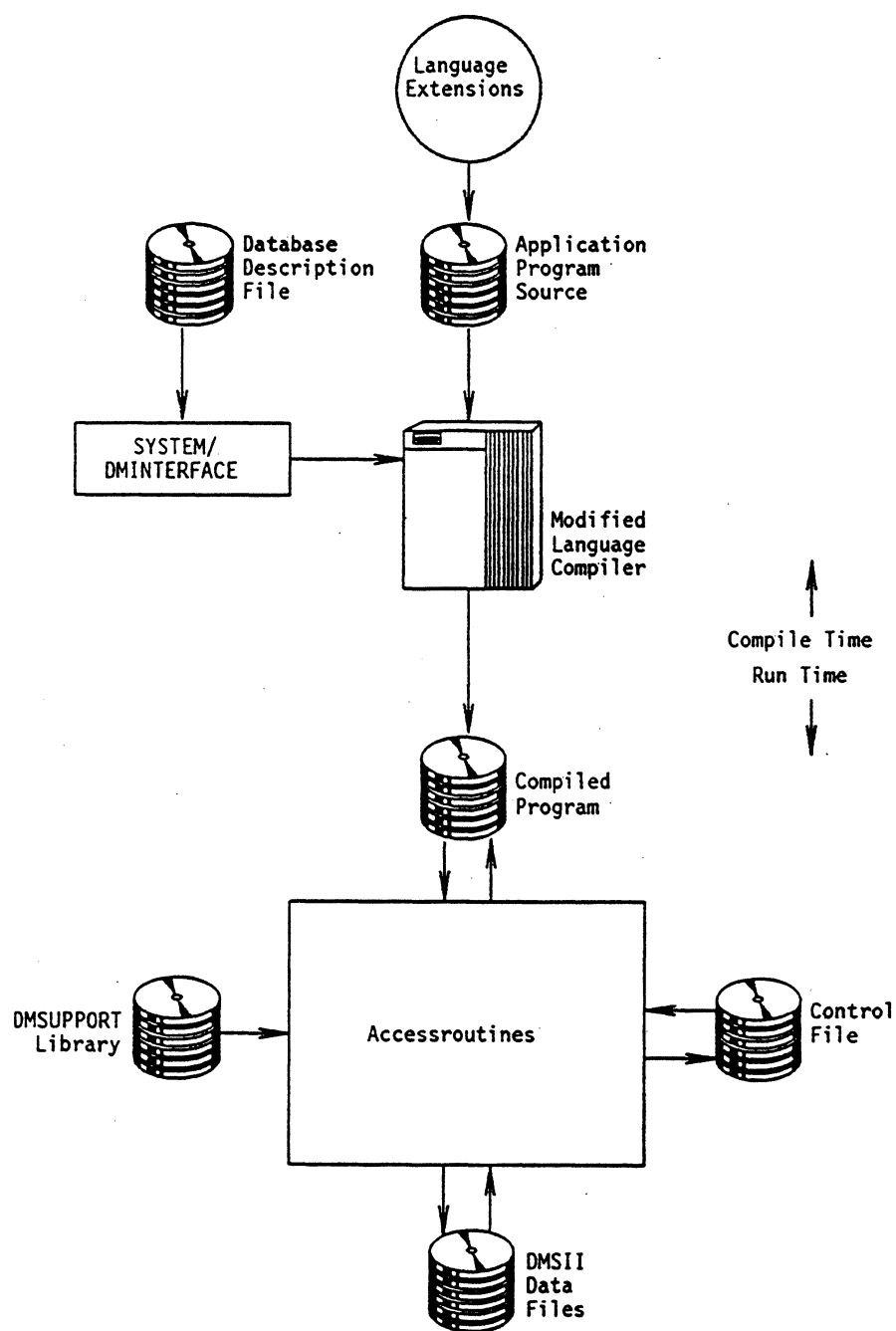


Figure 1-1. Database Processing with Language Extensions

Using SYSTEM/DMINTERFACE at Compile Time

The compile-time interface, SYSTEM/DMINTERFACE, is a separately compiled coroutine. It is responsible for giving information about the database and the language extensions used by the program to the user language compiler. The coroutine is needed because the standard language compilers have been extended to accept language extensions.

When a database is invoked, SYSTEM/DMINTERFACE searches for the description file of the specified database. The description file consists of tables that contain all the structure characteristics of the database. Every database has its own description file called DESCRIPTION/< database name > .

The system uses the description file as the central source of information when it compiles tailored software and user programs that access a particular database. The description file provides the compiler with a complete description of the database, including the record layout of each data set.

SYSTEM/DMINTERFACE performs the following tasks:

- Verifies the syntax of the database access operations
- Verifies that the data management operations are valid for the selected structure type
- Supplies the compiler with a description of those portions of the database selected by the program by using the database description file

If SYSTEM/DMINTERFACE finds any error, it reports the error to the user language compiler. If it does not find any errors, the coroutine generates calls on the appropriate procedures in the Accessroutines.

Using the Accessroutines at Run Time

When a program is successfully compiled, SYSTEM/DMINTERFACE generates calls to the Accessroutines. These calls are collectively referred to as the run-time interface. The language extensions also call on the DMSUPPORT library for error handling procedures.

The Accessroutines, regardless of the program interface, performs the same tasks. Read "Tasks Performed by the Accessroutines" in this section for an overview of the Accessroutines tasks.

Understanding the Basics of the Interpretive Interface

The interpretive interface uses the standard language compiler as its compile-time interface. The run-time interface, the Accessroutines, communicates with the database. This arrangement enables you to write and compile application programs before the database is compiled; the application program never directly accesses the database.

The following list gives the advantages of using the interpretive interface to access a database:

- Programs do not have to contain DMSII language extensions. The language must, however, support a library interface.
- A single application program, through the library mechanism, can invoke as many databases as desired. (A DMINTERPRETER library must be compiled for each database.)
- The open operation can be used to create general-purpose programs that can access any database.
- Decisions about database operations can be postponed until run time.
- Find operation enhancements optimize search techniques.
- Programs can use the intrinsic data set DBSTRUCTURE to interrogate the structure of a database at run time.

Figure 1-2 shows the modules and timing involved in compiling an application program that uses the interpretive interface.

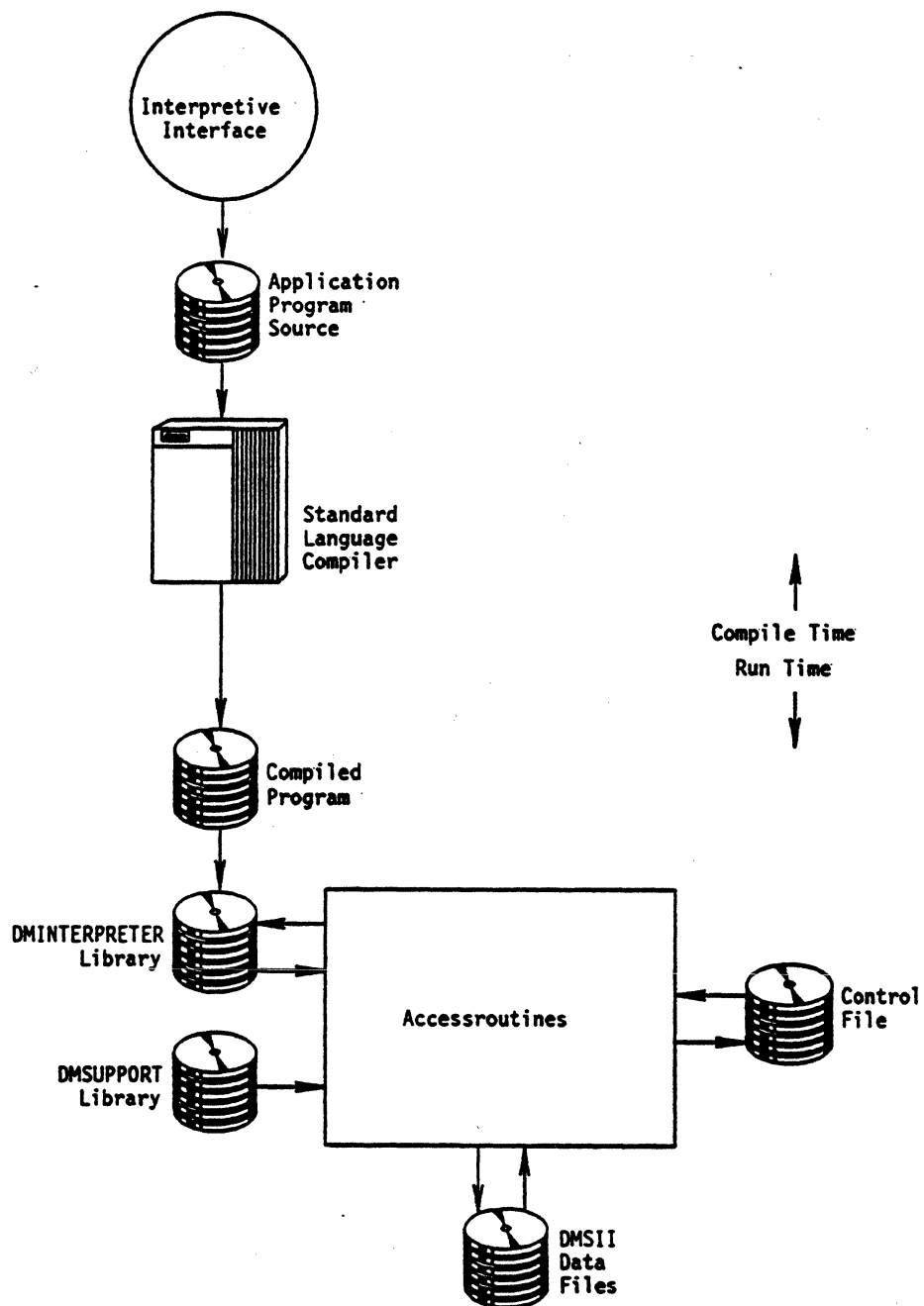


Figure 1-2. Database Processing with the Interpretive Interface

Using the Standard Language Compiler at Compile Time

The compile-time interface is the standard language compiler. The compiler is responsible for producing an error-free object code file for the program. It treats all errors as standard compiler errors.

Using the Accessroutines at Run Time

When a program is successfully compiled, the DMINTERPRETER library supplies the program with a description of the portions of the database selected by the program. The interpretive interface also calls on the DMSUPPORT library for error handling procedures.

The DMINTERPRETER library, in turn, makes calls to the Accessroutines. The Accessroutines returns results to the DMINTERPRETER library. The calls on the DMINTERPRETER library and the Accessroutines are collectively referred to as the run-time interface.

The Accessroutines, regardless of the program interface, performs the same tasks.

Tasks Performed by the Accessroutines

The Accessroutines enables many users to access the database concurrently. The Accessroutines is responsible for all physical and logical management of the database.

In a multiprocessing environment, the first program to open a database causes the Accessroutines to be loaded into main memory. The Accessroutines is *reentrant*, which means one copy of the code is stored in main memory and is shared by all users, regardless of what database they access.

Some tasks of the Accessroutines are to

- Verify the compatibility of a program with the Accessroutines.
- Perform a security check to make sure the program has permission to access the database.
- Perform a validity check on the data files in the database.
- Verify the timestamp for the data files.
- Decide whether the database is ready for access and, if the database is not, identify what recovery is required.

Section 2

Understanding Basic DMSII Programming Terms and Concepts

As a data management system, DMSII provides

- Data independence
- Data integrity
- Concurrent use of the data by more than one user or program
- Automatic recovery mechanisms

Before you begin writing application programs for a DMSII database, you should understand the basics of a data management system and how working with a database affects your program. This section builds on information in the DMSII Technical Overview to explain the reasons DMSII

- Separates data definition from data manipulation
- Distinguishes audited from unaudited databases
- Uses current paths and current records
- Locks records
- Re-creates records

Separating Data Definition from Data Manipulation

Conventional application programs describe the data to be managed, give instructions for inputting and extracting values, and store the values in discrete, program-dependent data files. If the data description changes, or if the method or location in which the data values are stored changes, the application program has to be changed. And, since each application has a separate data file, two or more versions of the same data could exist. This data redundancy can lead to inaccurate data.

In contrast, application programs using a data management system do not describe or define the structure of the data, nor do they give instructions concerning where to store the values. The application programs are responsible for adding, changing, deleting, and reporting data values only. The data management system is responsible for maintaining a common repository of data (called a *database*) and for connecting the program to a common data definition file (called a *description file*).

Using DMSII to Separate Data Definition from Data Manipulation

DMSII separates the task of defining data from the task of storing and manipulating data values. The structure of the data, its format and relationships, is maintained and

stored separately from the actual data values. The definition of the database structure is known as the *database schema*. Before any data value is entered into the database, the management system tests the value to ensure that it conforms to the definition. This is known as *data validation*.

Separating the tasks enables you to regulate the programs that change the data definition. Subtle changes can be made to the structure, such as adding a field, without recompiling the application programs.

The separation also means that a variety of application programs can access and change the data values without changing the structure and without causing data redundancy. Each application program can use all or part of the defined database without creating its own, separate set of data files, thus saving time and programming effort. Application programs can access information by referring to the structure, without knowing its physical location.

Because multiple programs can use the same database definition and values concurrently, the data management system must be able to manage both the database definition and values in a multiprocessing environment. The DMSII software, as described in the DMSII Technical Overview and the DMSII Utilities Operations Guide, is designed to handle all data management tasks. Your application can concentrate on the logical aspects of database organization and on updating or reporting information.

Defining the Database Schema with DASDL

A DMSII database is a collection of data in which the relationship among the data has been defined and stored in a description file.

A database administrator defines the database schema using the Data and Structure Definition Language (DASDL). DASDL is a high-level language that can specify the physical and logical characteristics of a database, as well as the criteria used to ensure the integrity and security of the data in the database.

The schema can be entered by directly coding a DASDL source file or by interactively using the Advanced Data Dictionary System (ADDS). Both methods create a description file for the database.

Consult the DMSII DASDL Reference Manual for details about DASDL. Read the *A Series InfoExec™ Administration Guide* for details about using ADDS with DMSII databases.

Creating the Database Schema

Through DASDL, each item is assigned a specific format, such as a length and character type. Data items are grouped together to create fixed hierarchical and network relationships. Features and options are established that govern the entire database or

individual groupings. For example, the database can be audited or unaudited. Each set, subset, and data set can have its own type.

The schema can consist of multiple data sets. Each data set contains data records. The data records are divided into items. For each data set, the database administrator can define a variety of sets and subsets to serve as indexes into the data set.

Using the Description File

Each database has one description file. The description file contains a complete description of the database. It is created when the DASDL source file is compiled.

The description file provides the application language compiler with the record layout of each data set. The system uses the description file when it compiles tailored software and user programs that access that database.

Changing the Database Schema

Changing the database definition can include any or all of the following tasks:

- Adding, changing, or deleting items defined for the records in a data set
- Adding or deleting data sets or sets
- Changing database attributes

Note: If any application program directly accesses data items that have been added or deleted during a reorganization of the database, the program must be changed and recompiled.

Monitoring Database Operations

The separation of the description from the data enables the database administrator to monitor and manage the database structure without affecting the actual data values. Table 2-1 lists the names of the available programs and their primary purpose. These programs can be run directly or through the Operations Control Manager (OCM) menu interface.

Table 2-1. Database Monitoring Programs

Program	Purpose
DBANALYZER	Analyzes logical and physical structure of a database. This program describes the structures in the database and their relationships, analyzes the database files, and tells how the files are accessed.
DBCERTIFICATION	Determines physical, internal, intrafile, and interstructure integrity of the database.

continued

Table 2-1. Database Monitoring Programs (cont.)

Program	Purpose
DMMONITOR	Interactively provides database status and statistics, and permits changes to database options and parameters.
DMUTILITY	Controls operational aspects of a database, such as dumping and loading database files and initializing structures.
VISIBLE DBS	Controls system resources associated with the database, such as allocating buffer core and determining status of individual tasks.

Read the DMSII Utilities Operations Guide for more information on DBANALYZER, DBCERTIFICATION, DMMONITOR, and DMUTILITY. The *A Series InfoExec Operations Guide* explains OCM. Consult the DMSII DASDL Reference Manual for a discussion of VISIBLE DBS parameters; how to use VISIBLE DBS is discussed in the DMSII Utilities Operations Guide.

Manipulating Data Values in Application Programs

A DMSII application program can

- Add values and new records to a database.
- Change existing values in the database.
- Delete values and records from the database.
- Report the current values stored in the database.

DMSII can accept data from both online and batch application programs. Each data management operation can be treated as a separate operation, or several operations can be grouped together and treated as one operation.

Invoking and Opening the Database

Although DMSII maintains a separate description file and data file for the database, your application program must use both files to access data values. Your program needs to know the structure and format to locate the data.

To meet this need, DMSII has a two-layered process for accessing data values. Your application program first must invoke the database and then must open the database.

- When you compile your program, invoking the database allows your program to use the structure and format of the database and records. (Invoking the database is also known as declaring the database.)
- When you run your program, opening the database connects your program to the Accessroutines.

Investigating the Database Schema

In many instances, the person who designs and maintains the database schema and the person who writes application programs for the database are not the same person. However, since the database schema limits the valid programming processes, before programming an application, review the database schema and do the following:

- Note what structures (such as data sets, sets, subsets, links, Accesses, logical databases) have been created. Which can you use in your program? What are the key items? Are duplicate key items allowed? In what order are they stored? What is the name and data type for each item?
- Find out what DASDL options have been set and what each option does. If your database is audited, check to see if `AUDIT`, `BACKOUTTOSPT`, `INDEPENDENTTRANS`, `REAPPLYCOMPLETED`, `RDSSTORE` or `KEYCHANGEOK` is set. Find the name and characteristics of the restart data set.
- Identify global data items.

Distinguishing between Audited and Unaudited Databases

DMSII supports two types of databases: audited and unaudited. The type of a database is set as part of the DASDL description for the database.

If a database is defined as audited, DMSII maintains a log of changes to the database. The log, known as the audit trail, contains the beforeimages and afterimages of every successfully completed update operation. DMSII does not maintain an audit trail for unaudited databases.

Note: A beforeimage is an image of the database record before any changes are made. An afterimage is an image of the database record after any changes are made and stored.

The type of database determines

- How updates are processed
- How the database is recovered

Processing Transactions

A *transaction* is an event that physically changes (updates) the database, such as storing a new record, storing a new value in an existing record, or deleting a record. *Transaction processing* refers to how and when the software handles a transaction.

- For an audited database, a series of transactions are grouped into a single, logical transaction.
- For an unaudited database, each update is treated as a separate and distinct transaction.

Processing Transactions in Audited Databases

For audited databases, DMSII software allows an application program to logically group update operations and to process the group as a single transaction. The start of the

group transaction is signaled by a begin transaction operation. The end of the group transaction is identified by an end transaction operation. While the program is between a begin and end transaction operation, the program and the database are in *transaction state*.

Any operation that physically changes the content of the database should be performed only when the application program is in transaction state. These operations are assign, delete, generate, insert, remove, and store.

Figure 2-1 illustrates a logical transaction in transaction state.

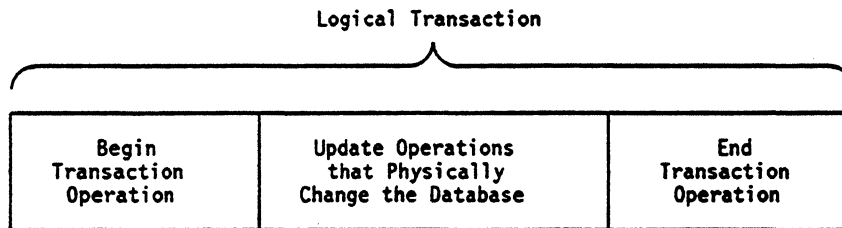


Figure 2-1. Transaction State

Transaction Processing Guidelines for Audited Databases

Because of the importance of monitoring and controlling changes, transaction state is a crucial time for an update program. The following guidelines apply when updating an audited database:

- All operations that physically change the database must be performed in transaction state.
- The program that initiates the begin transaction operation must be the one to execute the end transaction operation.
- Logical transactions cannot be nested. Programs cannot process begin transaction operations when they are in transaction state. End each logical transaction before processing another logical transaction.
- Do not issue any go-to instructions in transaction state. Keep transaction processing as linear as possible.
- Do not perform any operation that causes the system to remain in transaction state any longer than absolutely necessary. For example, do not perform any sort operations or read a remote file in transaction state.

Processing Transactions in Unaudited Databases

For unaudited databases, each transaction is a separate operation. DMSII processes each operation as it is executed.

Recovering a Database

Each time processing is halted abnormally, the database must be recovered. *Recovery* means reconstructing the database, ensuring that no partially completed transactions have been applied to the database, and notifying programs where to restart processing.

- For an audited database, DMSII maintains information about each change to the database in a restart data set and an audit trail. This information is used by DMSII to recover the database automatically.
- An unaudited database does not use DMSII features to maintain a record of changes. Recovery of an unaudited database must be performed manually.

Recovering Audited Databases

Because changes to an audited database are managed by DMSII software, an audited database is used normally when either or both of the following are true:

- Multiple application programs access the same database.
- Application programs perform online operations.

In both cases, DMSII governs the traffic pattern of how and when programs access and update the database to prevent data corruption and concurrency problems.

To manage the traffic pattern, the DMSII software maintains a log of changes using both a restart data set and an audit trail.

- A *restart data set* contains information that points to the last completed transaction that updated the database. This information enables the program to restart from a specific point.
- The *audit trail* contains the beforeimages and afterimages for changed, added, and deleted records. This information is used during recovery from a hardware or software failure to re-create the database.

The restart data set and the audit trail are used to recover the database; audited databases are rarely restored manually. The DASDL definition of an audited database must contain a restart data set and must set the AUDIT option. In addition, the DASDL code can have an audit trail declaration and can set the following four options: BACKOUTTOSPT, INDEPENDENTTRANS, REAPPLYCOMPLETED, and RDSSTORE. In addition, your program logic can contain recovery-specific data management operations that determine where your program should restart after a failure.

For guidelines on the effect of the DASDL options and how to use DMSII recovery operations, read Section 8, "Recovering an Audited Database." Consult the DMSII DASDL Reference Manual for detailed information about the audit trail declaration, the restart data set, and recovery options defined through DASDL.

Recovering Unaudited Databases

Unaudited databases are normally used for simple batch operations by single-purpose application programs.

If the application program has a failure or it is discontinued, the Accessroutines usually can close the database. However, the database is left in a partially updated state—some changes made by the application program have been applied to the database and others have not. Unless your program can restart at a specified point, it is better to restore the database manually and rerun the program.

Since the primary means of recovering an unaudited database is restoring the database from a tape (also known as dumping a tape or manually restoring), it is important to maintain current and accurate tapes for database recovery.

Understanding Current Path and Current Record

On a program-by-program basis, DMSII software keeps track of the data structures that are currently invoked and the records that are currently in use.

Determining What Structures Are Available to a Program

When your program invokes the database, the program restricts what data structures can be accessed.

- If your program invokes the *physical database*, the program can access all the data sets, sets, and subsets defined in the description file for that database.
- If your program invokes a *logical database*, the program can access structures defined for that logical view only.

Your program can access the actual structures and their values only after the program opens the database.

Relating the Current Path to the Current Record

As your program invokes a structure, the DMSII software creates and maintains a pointer for that structure. This pointer, known as the *current path*, points to the record in the structure your program is currently using. The record is known as the *current record*. The current path and the current record are used by data management operations and the Accessroutines to locate a record. Figure 2-2 shows the relationship of the current path to the current record.

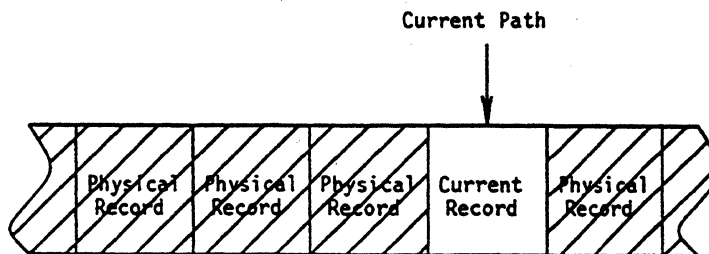


Figure 2-2. Current Path and Current Record

If multiple programs access the same structure, DMSII maintains a separate current path for each program. Each current path points to the current record for that program.

Note: *Each invoked structure has a user work area, also known as a work area or record area. This is the area in the application program where data set records are constructed, accessed, or modified. In each application program, one user work area is maintained for each invoked data set.*

Identifying the Beginning and End of a Structure

A structure consists of physical records. The first time the structure is used in a data management operation, the current path points to one record position before the first physical record in the file. This position is known as the beginning of the structure.

As records are accessed and updated, the current path is maintained by the Accessroutines. Each time you execute a set operation or a search operation, the Accessroutines repositions the current path. (A search operation is any delete, find, lock, modify, or secure operation.)

The current path can be positioned anywhere between the beginning of the structure and the end of the structure. The end of the structure is one record position after the last record in the file.

Figure 2-3 shows the relationship of the beginning and end of a structure to the physical records.

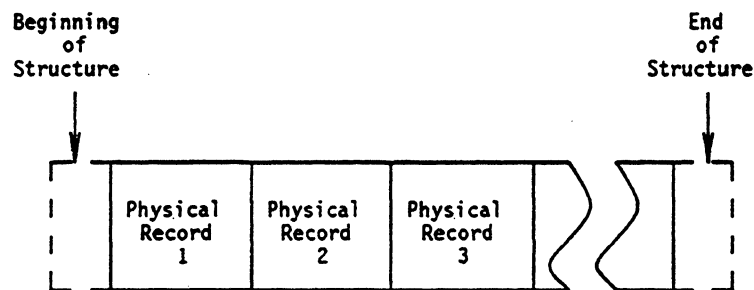


Figure 2-3. Beginning and End of a Structure

Maintaining Separate Paths for Each Structure

The first time you use a structure in your application program, DMSII creates a unique current path that points to the beginning of the structure. The structure can be a subset, set, or data set. The path is valid only for your program and that invocation of the structure.

For instance, as shown in Figure 2-4, if the database has three data sets (DS1, DS2, and DS3), and two subsets (SS1 and SS2), and your program invokes each of these structures once, DMSII creates, monitors, and maintains five unique paths.

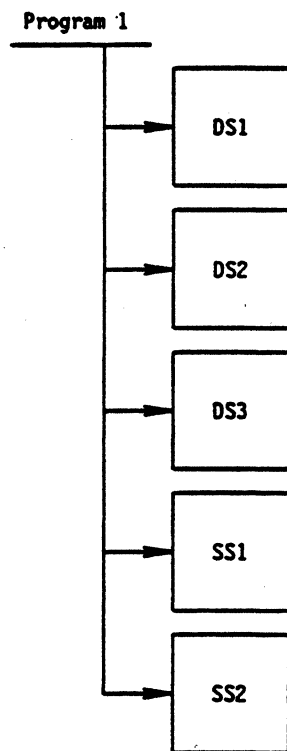


Figure 2-4. Multiple Paths in One Application Program (One for Each Structure)

If the same program uses the same set more than once, each use has its own path. These multiple paths can be seen in Figure 2-5; Program 1 uses SS1 twice.

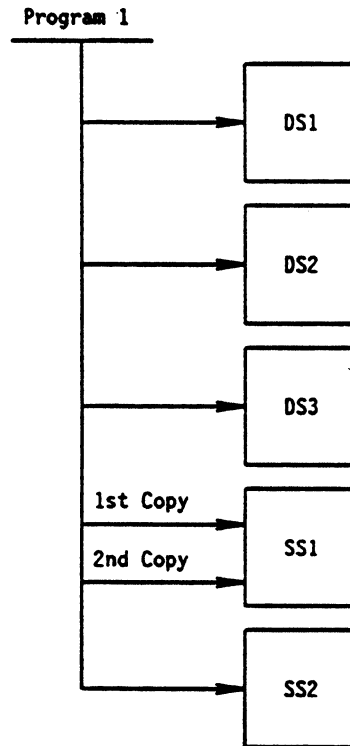


Figure 2-5. Multiple Uses of the Same Structure in One Application Program

And, if more than one program concurrently opens the same database, DMSII creates and maintains distinct paths for each invocation in each program. In Figure 2-6, Program 1 uses DS1 and DS2. Program 2 uses DS2 only. And, Program 3 uses DS2 and SS1 and SS2. DMSII creates, monitors, and maintains the six unique current paths.

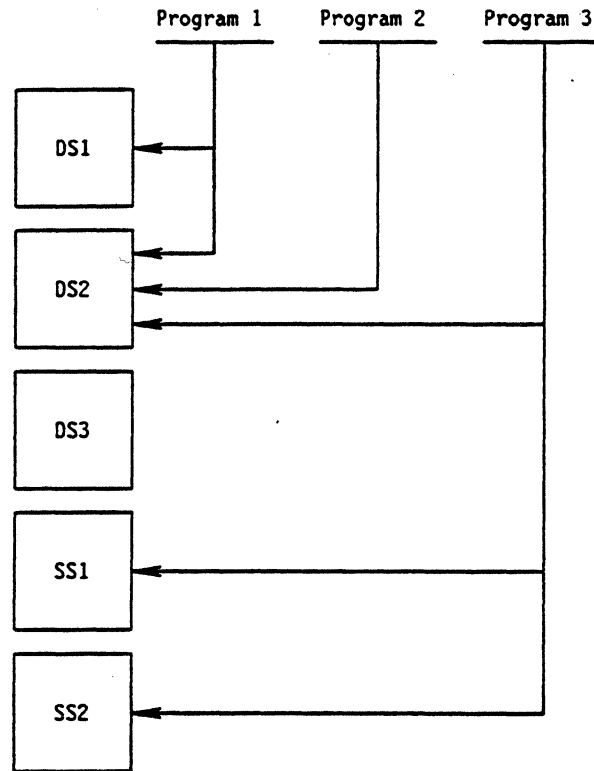


Figure 2-6. Current Paths When Multiple Application Programs Use the Same Structures

Note: Paths are user oriented. If one user or one program performs a search operation on a set, it does not affect the path of any other invocation of that set.

Defining the Status of the Current Path

A current path can be either valid or invalid.

- A valid current path for a data set refers to a location that contains a valid structure, such as a record or a data item.
- A valid current path for a set or subset points to a location in the set or subset that contains a key item. (A *key* is the data item on which records of a set or subset are ordered and retrieved.)

- An invalid path points to a *virtual record*—a nonexistent record. An invalid path occurs when the current path points to
 - The beginning of a structure
 - The end of a structure
 - A deleted record
 - The logical space between two existing records.

You can correct an invalid current path by using the set operation or a find operation to reposition the pointer. Read Section 5, “Setting the Current Path,” for guidelines on how to correct an invalid current path.

Example of Logical Space Between Two Existing Records

In Figure 2-7, records are stored in ascending, alphabetical order; the first record is for Merino and the last record is for Quinton.



LAST NAME	ID	BLOOD-TYPE
Merino	731	O
Naverin	692	A
Shaw	927	AB
Quinton	548	AB

Figure 2-7. Space Between Records

If your program requested records beginning with “R,” the path would point to the logical space between Naverin and Shaw. The space between the records is not a physical space; it is the logical space between letters in the alphabet.

Defining the Status of the Current Record

The status of the current record reflects the status of the current path.

- The current record is considered *defined* when the current path is valid.
- The current record is considered *undefined* when the current path is invalid. For example, when a structure is first opened, the current record is undefined because the current path points to the beginning of the structure.

Locking and Securing a Record

One advantage of DMSII is the ability for many programs and many users to access the same database at the same time. For instance, if one database contains all the inventory information, the accounting department can have five users updating the price of parts at the same time that eight users in the receiving department are updating the number of parts in stock.

However, along with this advantage comes the task of controlling the updates to prevent two or more users from accessing and changing the same record at the same time. The DMSII application program interface supports several locking operations that control whether a record can be concurrently retrieved or updated.

- A lock, modify, or delete operation guarantees a user exclusive use of a record until the user frees (unlocks) the record. No other user or program can access or change the record until the record is freed.
- A secure operation guarantees no other user or program can update the record until the user frees the record. Other users and programs can access the record to retrieve information.

Note: A find operation does not lock or secure a retrieved record.

If you need to update a record, lock the record. If you want to read a record, secure it.

Freeing a Record

Whenever you lock or secure a record, make certain your program code frees the record. Freeing a record releases the record from all lock or secure restrictions. The record is available then to other users and programs. How and when you free a record depends on whether the record was locked or secured outside or inside transaction state.

If a Record is Locked Outside Transaction State

If your program locks or secures a record outside of transaction state, the program can implicitly free the record outside of transaction state by executing a create, find, lock, modify, recreate, or secure operation for another record in the same structure. Or, your program can explicitly free the record by executing a free operation.

If a Record is Locked Within Transaction State

When your program locks or secures a record outside transaction state and then enters transaction state, or when your program locks or secures a record within transaction state,

- If concurrency control is set for the database through the DASDL *INDEPENDENTTRANS* option, DMSII automatically frees any locked or secured record when it successfully executes an end transaction operation. You cannot implicitly or explicitly free the record.
- If no concurrency control is set for the database, you can free the record by executing a create, delete, find, free, lock, modify, recreate, secure, or store operation within that transaction state. DMSII automatically frees any locked or secured record when it successfully executes an end transaction operation.

Read "Setting the DASDL INDEPENDENTTRANS Option for Concurrency Control" in Section 8.

Locking, Securing, and Freeing a Structure

If the *DASDL INDEPENDENTTRANS* option is set for your database, you can lock or secure an entire structure, such as a data set. Locking or securing a structure allows your program to have exclusive use of the structure. This is useful if your program needs to update many records in the same data set.

To release a structure from a lock or secure operation, your program must execute an explicit free operation or close the database. A halt/load recovery also frees the structure.

Creating and Re-creating a Record in an Application Program

Both the create and recreate operations are used with data transfer operations and the store operation to add records to a database.

- Use the create operation when you want to add a new record.
- Use the recreate operation when you want to change a required item and duplicate key items are not allowed.

Creating a record in an application program places the record definition in the user work area. Each item in the record is initialized to its default initial value. Your program code can then place values into the record. You must issue a store operation to add the record to the database.

Re-creating a record places a duplicate of the last accessed record in the user work area, including the data values. (Usually the delete operation precedes the recreate operation; an image of the deleted record is the basis of the duplicate record.) After changing the required item, you can use the values in the original record in the new record, or you can change the values. You must issue a store operation to add the record to the database.

If the *DASDL INDEPENDENTTRANS* option is set for the database and the *DASDL KEYCHANGEOK* option is set for the structure, you do not have to use the recreate operation. Below is a comparison of the programming logic flow when the options are set and when they are not set.

Options Set	Options Not Set
Lock record	Lock record
Make changes	Delete record
Store record	Re-create record
	Make changes
	Store record

Read Section 7, "Updating the Database," for information about when and how to use the create and recreate operations.

Section 3

Identifying Data Management Tasks

Application program data management tasks can be divided into the following major areas:

- Invoking, opening, and closing a database
- Finding information in a database
- Transferring data
- Adding, storing, and deleting records
- Handling errors and exceptions

If your application program manipulates data in an audited database, the program must include the basic transaction management instructions—begin transaction and end transaction. By using language extensions, the program can use more detailed transaction management operations. In addition, any application program that accesses an audited database should include recovery instructions.

Each task can involve one or more data management operations. Table 3-1 lists the tasks. Within each task, the operations are given in alphabetical order.

Table 3–1. Data Management Operations Listed by Task

Task	Operations
Invoking, opening, and closing a database	Close
	Invoke/declare
	Open
Finding information	Find
	Free
	Lock
	Modify
	Secure
	Set (current path)
Transferring data	Assign
	Compute
	Get
	Put

continued

Table 3-1. Data Management Operations Listed by Task (cont.)

Task	Operations
Adding, storing, deleting records	Create
	Delete
	Generate
	Insert
	Recreate
	Remove
	Store
Handling errors and exceptions	DMKEY
	DMTERMINATE
Managing transactions	Abort transaction
	Begin transaction
	Cancel transaction point
	End transaction
	Savepoint

The availability of an operation and the actual code to perform these tasks varies with the programming language and with the application interface (language extensions or the interpretive interface entry points).

For audited databases, data management operations that change the physical database—assign, delete, generate, insert, remove, and store—must be performed in transaction state. Transaction state is initiated by a begin transaction operation and ended by an end transaction operation.

For unaudited databases, data management operations that change the physical database can be performed at any time.

Dividing Tasks by Interface and Type of Database

Table 3-2 lists the data management operations in the order that the operation is most likely to be performed. For each operation, the table shows

- Whether it can be run from the interpretive interface entry points or from the language extensions
- Whether it applies to an audited or an unaudited database

Table 3-2. Application Program Data Management Tasks

Task	Entry Points	Language Extensions	Audited Database	Unaudited Database
Invoking a database	Yes	Yes	Yes	Yes
Opening a database	Yes	Yes	Yes	Yes
Setting the current path	Yes	Yes	Yes	Yes
Creating new records	Yes	Yes	Yes	Yes
Finding records	Yes	Yes	Yes	Yes
Beginning transaction state	Yes	Yes	Yes	No
Locking records	Yes	Yes	Yes	Yes
Transferring data	Yes	Yes	Yes	Yes
Setting intermediate transaction points	No	Yes	Yes	No
Deleting records	Yes	Yes	Yes	Yes
Re-creating records	Yes	Yes	Yes	Yes
Storing records	Yes	Yes	Yes	Yes
Aborting transactions	No	Yes	Yes	No
Canceling transactions	No	Yes	Yes	No
Ending transaction state	Yes	Yes	Yes	No
Examining status of last database operation	Yes	Yes	Yes	Yes
Freeing records	Yes	Yes	Yes	Yes
Closing a database	Yes	Yes	Yes	Yes
Terminating the program	Yes	Yes	Yes	Yes

Identifying Interpretive Interface Data Management Operations

To perform data management functions, ALGOL, COBOL, COBOL74, COBOL85, and FORTRAN77 programs can use entry points. Tables 3-3 through 3-5 list the interpretive interface entry points associated with standard database operations, data transfer operations, and exception handling operations, respectively.

Consult the DMSII Interpretive Interface Programming Manual for details about entry point functions and language-specific syntax.

Standard Entry Point Operations

Table 3-3 lists the standard entry points of the interpretive interface by task. Each entry point name is language specific. These entry points perform standard data management operations.

Table 3-3. Standard Entry Points by Task

Task	Entry Points
Opening a database	ALGOOPEN DBOPEN FORTRAN77OPEN
Creating new records	ALGOLCREATE DBCREATE FORTRAN77CREATE
Setting the current path	ALGOLSET DBSET FORTRAN77SET
Freeing records	ALGOLFREE DBFREE FORTRAN77FREE
Finding records	ALGOLFIND DBFIND FORTRAN77FIND
Beginning transaction state	ALGOLBEGINTRANSACTION DBBEGINTRANSACTION FORTRAN77BEGINTRANSACTION
Locking records	ALGOLLOCK DBLOCK FORTRAN77LOCK
Deleting records	ALGOLDELETE DBDELETE FORTRAN77DELETE
Re-creating records	ALGOLRECREATE DBRECREATE FORTRAN77RECREATE

continued

Table 3–3. Standard Entry Points by Task (cont.)

Task	Entry Points
Storing records	ALGOLSTORE
	DBSTORE
	FORTRAN77STORE
Ending transaction state	ALGOENDTRANSACTION
	DBENDTRANSACTION
	FORTRAN77ENDTRANSACTION
Closing a database	ALGOLCLOSE
	DBCLOSE
	FORTRAN77CLOSE
Executing language extensions	ALGOLVERB
	DBVERB
	FORTRAN77VERB

Data Transfer Entry Point Operations

Table 3-4 lists the data transfer entry points of the interpretive interface by task. Each entry point name is language specific. The DBGET entry points move data from the user work area to user-declared variables in your application program. The DBPUT entry points move data from variables in your application program to the user work area.

Table 3–4. Data Transfer Entry Points by Task

Task	Entry Points
Moving a character string to a variable	ALGOLGETSTRING
	DBGETDISPLAY
	FORTRAN77GETCHARACTER
Moving a Kanji character string to a variable	ALGOLGETKANJI
	DBGETKANJI
	FORTRAN77GETKANJI

continued

Table 3–4. Data Transfer Entry Points by Task (cont.)

Task	Entry Points
Moving a numeric value to a variable	ALGOLGETREAL DBGETREAL FORTRAN77GETREAL
Moving a double-precision value to a variable	ALGOLGETDOUBLE DBGETDOUBLE FORTRAN77GETDOUBLE
Retrieving Boolean values	ALGOLGETBOOLEAN DBGETBOOLEAN FORTRAN77GETLOGICAL
Placing a string into a data item	ALGOLPUTSTRING DBPUTDISPLAY FORTRAN77PUTCHARACTER
Placing a string into a Kanji item	ALGOLPUTKANJI DBPUTKANJI FORTRAN77PUTKANJI
Placing a numeric value into a data item	ALGOLPUTREAL DBPUTREAL FORTRAN77PUTCHARACTER
Placing a double-precision value into a data item	ALGOLPUTDOUBLE DBPUTDOUBLE FORTRAN77PUTDOUBLE
Setting a data item to a Boolean value	ALGOLPUTBOOLEAN DBPUTBOOLEAN FORTRAN77PUTLOGICAL
Setting a data item to a null value	ALGOLPUTNUL DBPUTNULL FORTRAN77PUTNULL
Constructing transfers during program execution	ALGOLDATA DBDATA FORTRAN77DATA

Exception Handling Entry Point Operations

Table 3-5 lists the interpretive interface entry points used to report exceptions and errors. Each entry point name is language specific.

Table 3-5. Exception Handling Entry Points by Task

Task	Entry Points
Returning the exception word	ALGOLSTATUS DBSTATUS FORTRAN77STATUS
Returning the text of an exception message	ALGOLEXCEPTIONTEXT DBEXCEPTIONTEXT FORTRAN77EXCEPTIONTEXT
Identifying the type of exception	ALGOLEXCEPTIONNAME DBEXCEPTIONNAME FORTRAN77EXCEPTIONNAME

Using Language Extensions as Data Management Statements and Commands

To perform data management functions, the ALGOL, COBOL, COBOL74, COBOL85, and RPG compilers have been extended to support data management operations. Each extension is a data management statement or command. These operations are listed alphabetically in Table 3-6, each with a brief description. Consult the specific language reference manual for a detailed explanation of the statements and commands.

Notes:

- *Not all operations are supported in all languages. Each language uses a subset of the available data management operations. Check the specific language reference manual for a listing of supported operations.*
- *The interpretive interface initiates many of the data management operations through the ALGOLVERB, DBVERB, and FORTRAN77VERB standard entry points.*

Table 3–6. DMSII Language Extension Operations

Statement or Command	Purpose
ABORTTRANSACTION	Discards any updates made in a transaction after a BEGINTRANSACTION or DMSAVEPOINT operation and removes the program from transaction state.
ASSIGN	Establishes a link from one data set record to another.
BEGINTRANSACTION	Places a program in transaction state.
CANCEL TRANSACTION POINT	Discards all updates made in a transaction back to an intermediate transaction point (a savepoint) or back to the beginning of the transaction.
CLOSE	Closes a database when further access is no longer required.
COMPUTE	Assigns a value to a Boolean item in the current record of a data set.
CREATE	Initializes the user work area of a data set record.
DELETE	Removes a record from a data set, and removes its reference from all sets and automatic subsets.
DMKEY	Provides a method of specifying the conditions that must be satisfied by the record accessed in the previous find, lock, or delete operations.
DMSAVEPOINT	Writes a savepoint record to the audit trail.
DMTERMINATE	Terminates the program. When an exception occurs that the program does not handle, the program is terminated with a fault.
ENDTRANSACTION	Takes a program out of transaction state.
FIND	Transfers a record to the user work area associated with a data set or global data. A find operation reads a record from a data set, even if the record is locked or secured. It can return a record that is part of an unapplied transaction to the program.
FREE	Explicitly unlocks the current record of a data set.
GENERATE	Creates a subset from one or more manual subsets.
GET	Moves values from database variables to program variables.
IF	Makes a conditional transfer of control based on the value of an item in a data set.
INSERT	Inserts a record into a manual subset.
LOCK	Performs the same function as a find operation, except that the record is locked against concurrent modification by another user.
MODIFY	Performs the same function as a lock operation.

continued

Table 3–6. DMSII Language Extension Operations (cont.)

Statement or Command	Purpose
OPEN	Opens a database for subsequent access, and specifies the access mode.
PUT	Moves values from program variables to database variables.
RECREATE	Partially initializes the user work area. Initializes only the data set, set, link, and control items of the record area of a data set. All data items remain unaltered.
REMOVE	Removes a record from a manual subset.
SECURE	Prevents any other program from updating a record. Multiple programs can secure the same record. The secure operation waits for a locked record to be freed before returning the record.
SET	Alters the current path or changes the value of an item in the current record.
STORE	Places a new or modified record in a data set or the global record area.

Section 4

Opening and Closing a Database

Before performing any other database operations, you must first explicitly invoke and then open the database.

Although ending your program implicitly closes the database, you can explicitly close the database for cleaner and clearer code. If your program opens an audited database for update, explicitly closing the database is a recovery safeguard.

Invoking a Database

Invoking a database

- Makes the names of data items available to the application program
- Informs the compiler to include special registers and syntax checking

Depending on the user language, the database is invoked through either an invocation or a declaration. For example, COBOL invokes a database. ALGOL and FORTRAN77 declare a database.

For ease of reading, the term *invoke* is used in this section to refer to both an invocation and a declaration.

Invoking either a Physical or a Logical Database

Your program can invoke either the total, physical database or a logical view of the database.

- Invoking a physical database makes the entire database available to the program.

Data in physical databases are not independent of the programs that use the data. The data is stored in data sets. When the record formats in a physical database are changed, all programs that use those data sets must be changed and recompiled.

- Invoking a logical database limits the available data sets, sets, subsets, and remaps available to the program. It also limits the activities the program can perform.

Logical databases, including remaps, represent only parts of a database; they provide a predetermined view of the database, enforce structure-level security, and offer data independence. Logical databases provide greater flexibility and reduce the time it takes to execute database open and close operations.

Invoking a logical database that uses remaps minimizes the need to recode and recompile application programs when changes are made to the database.

Overriding the Database Named in a Database Invocation

Using the WFL database equation, you can equate an invoked database to another database either at program compilation or at run time. Use the database equation in order to

- Take advantage of the reentrance capability of the Access routines.
- Access databases that are stored under other usercodes.
- Access databases on families that are not visible to a task.
- Access modeled databases.

You can use the following syntax either in a WFL job or through CANDE. The database name must be the same as the internal name declared in your program for the database.

< database equation >

```
— DATABASE —<database name>—————→  
← ( — TITLE — = —<database title>— ) —————→
```

Note: A run-time error results if you attempt to examine the TITLE attribute at run time and the database is open.

Read the Work Flow Guide for details of the task attributes and the WFL Reference Manual for a detailed explanation of the syntax. Consult the DMSII DASDL Reference Manual for a complete explanation of modeling.

Example of Using a WFL Database Equation

The following example illustrates the use of a WFL database equation during compilation and execution:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% During compilation, the database CTIMEDB is equated to the database %  
% TESTDB under a different usercode and pack. The database EQUATEDDB %  
% is equated to LIVEDB under the same usercode and pack. Whenever the %  
% program encounters CTIMEDB, it substitutes TESTDB. Whenever it %  
% encounters EQUATEDDB, it substitutes LIVEDB. %  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
?BEGIN JOB WFL/EXAMPLE;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% Overriding equation during compilation. %  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
COMPILE USER/PROGRAM WITH BDMSALGOL LIBRARY;
DATABASE CTIMEDB (TITLE = (UC)TESTDB ON TESTPACK);
COMPILER DATA
BEGIN % ALGOL SOURCE
```

```
    DATABASE CTIMEDB;
    DATABASE MYDB = RTIMEDB;
    DATABASE EQUATEDDB (TITLE = "LIVEDB");
    OPEN CTIMEDB;
```

```
    ...
```

```
    CLOSE CTIMEDB;
    OPEN MYDB;
```

```
    ...
```

```
    CLOSE MYDB;
    OPEN EQUATEDDB;
```

```
    ...
```

```
    CLOSE EQUATEDDB;
```

```
END.
```

```
?
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               End ALGOL input.                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% During execution, the database MYDB is equated to the database             %
% SOMEOTHERDB and EQUATEDDB is equated to HISDB under the usercode UC, %
% on pack THEIRPACK.                                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
RUN USER/PROGRAM;
    DATABASE MYDB (TITLE = SOMEOTHERDB);
    DATABASE EQUATEDDB (TITLE = (UC)HISDB ON THEIRPACK);
?END JOB.
```

Example of Using a WFL Database Equation through CANDE

The database equation can be executed through CANDE only in a WFL statement; for example:

```
WFL BEGIN JOB;
    RUN OBJECT/DB/PROGRAM;
        DATABASE LIVEDB (TITLE = (UC)TESTDB ON TESTPACK);
END JOB.
```

Sharing the Same Database Declaration

If multiple tasks share the same database declaration, they also share the same user work areas, user key areas, and current path. The programs must be coded so that one task does not cause any problems for other tasks.

One method of preventing problems is to use locks. You can have your program lock each structure, the user key area, and each transaction. Lock a structure as long as you need to guarantee that the path for that structure is not changed by another program. Lock the user key area while executing any DMSII operation that uses a key.

If the database contains global data and the invoked database contains a remap of the global data, use the name of the logical database—not the name of the global data remap—to find and lock the global data record.

Note: *If your program uses a database outside of a transaction processing system (such as the Communication Management System) or uses an unaudited database, do not attempt to share a database declaration.*

Opening a Database

The open operation establishes the way in which the program accesses data values and performs database operations.

- *Inquiry access* allows the program to retrieve data but denies permission to add, delete, or alter data.
- *Update access* allows the program to manipulate data (add, delete, or alter) and retrieve data.

Use inquiry access when you wish to only retrieve data. Use update access when you need to retrieve and manipulate data.

If you are using language extensions, you must specify the name of the database in every open operation. If the name was declared, you can use the actual name of the physical database or the name of the logical database.

If you are using the interpretive interface, the open operation entry point assumes that the database name is the name of the declared database. You can create an application program that can access any database.

Multiple databases can share one copy of the Accessroutines. If a database is already invoked and opened by the same or another program, subsequent open operations take less time. To reduce system overhead, open each database only once in any given program.

Open an audited database outside of transaction state. After opening the database, determine if there is a restart data set record for the program. If there is a restart record, restart processing from the point specified in the record. (Read Section 8, "Recovering an Audited Database," for more information on restart data sets and using the restart record.)

Closing a Database

A database is implicitly closed when the library execution is complete. However, explicitly closing the database makes your code cleaner and clearer.

If your program opens an audited database for update, explicitly closing the database is a recovery safeguard.

Note: *For audited databases, make sure that you have explicitly ended transaction state before closing the database. If you attempt to close a database while the program is in transaction state, the Accessroutines initiates the ABORT task.*

Section 5

Setting the Current Path

Several data management operations, shown in Table 5-1, can be used to set (change) the current path.

Table 5-1. Data Management Operations that Set the Current Path

Operations	Effect
Store, preceded by create or recreate	A create or recreate operation creates a record in the user work area only. A store operation adds the record to the structure. The current path is updated to point to the new record.
Delete, find, lock, modify, or secure	The current path points to the record retrieved by any of the search and retrieval operations. <i>Note: A search operation always starts from the current record. If this is the first time the program uses the structure, the current path points to the beginning of the structure. If your program has used the structure before, the current record is the last accessed record.</i>
Insert or remove	The current path points to the data set record referenced in an insert or remove operation.
Set	The set operation positions the current path to either the beginning (before the first record) or the end (after the last record) of the structure.

Whenever your program initiates a search for a record, there can be two outcomes: the record is found and the current path is changed, or the record is not found and a NOTFOUND exception is returned. This section describes how you can use the set and search operations to set the current path or reset an invalid current path. (An invalid current path points to a virtual record.)

Although the text in this section refers to the find operation, you can substitute any of the search operations. Read Appendix A, "DMSII Exceptions and Errors," for details about NOTFOUND exceptions.

Using a Set Operation

A set operation can be used to position the current path before initiating a search operation. A set operation is most useful in a serial search of the entire set or subset.

- A *set to beginning* command positions the current path to the beginning of the structure, before the first physical record.

Use a set to beginning operation before a series of find next operations. The current path is set to the beginning of the file. If a find next operation returns a NOTFOUND exception, you have scanned all records.

- A *set to ending* command positions the current path to the end of the structure, after the last physical record.

Use a set to ending operation before a series of find prior operations. The current path is set to the end of the file. If a find prior operation returns a NOTFOUND exception, you have scanned all records.

Using Search Operations to Set the Current Path for a Data Set

Any find operation that retrieves another record in a data set (not the current record), sets the current path to the retrieved record.

Examples of Setting the Current Path for a Data Set

Figure 5-1 shows the EMPLOYEES data set used in the following examples and illustrates the results of the find operations.

EMPLOYEES Data Set

Record Address	EMP-NAME	EMP #	EMP-ADDRESS	
1	Chin,David	233189	Sopchopopy,FL	Find Next Operation
2	Endicott,Race	372451	Empire,LA	
3	Garcia,Mary	132445	Buncombe,IL	Find Prior Operation
4	Johnson,Bill	441234	Nyack,NY	
5	Smith,John	121221	Perris,CA	Find Prior Operation
6	Warren,Aldo	178556	Cocolalla,ID	
7	Wells,Sheila	224534	Giddings,TX	

Figure 5-1. Resetting the Current Path for a Data Set

Example 1: Find Next

If the current path initially points to the beginning of the data set, a find next operation retrieves the first physical record in the data set (for David Chin).

Example 2: Find Prior

If the current path initially points to the end of the data set, a find prior operation retrieves the last physical record in the data set (for Sheila Wells).

Example 3: Find Prior with a Deleted Record

If the current path initially points to the fourth record in the data set (for Bill Johnson) and the record for Mary Garcia was deleted, a find prior operation (before any other changes are made) retrieves the record for Race Endicott.

Using Search Operations to Set the Current Path for a Set or Subset

Any find operation that retrieves another record in a data set using a set or subset, sets the current path to the retrieved record.

Examples of Setting the Current Path for a Set

Figure 5-2 shows the EMPLOYEES data set and the BY-NUMBER set used in the following examples and illustrates the results of the find operations.

Setting the Current Path

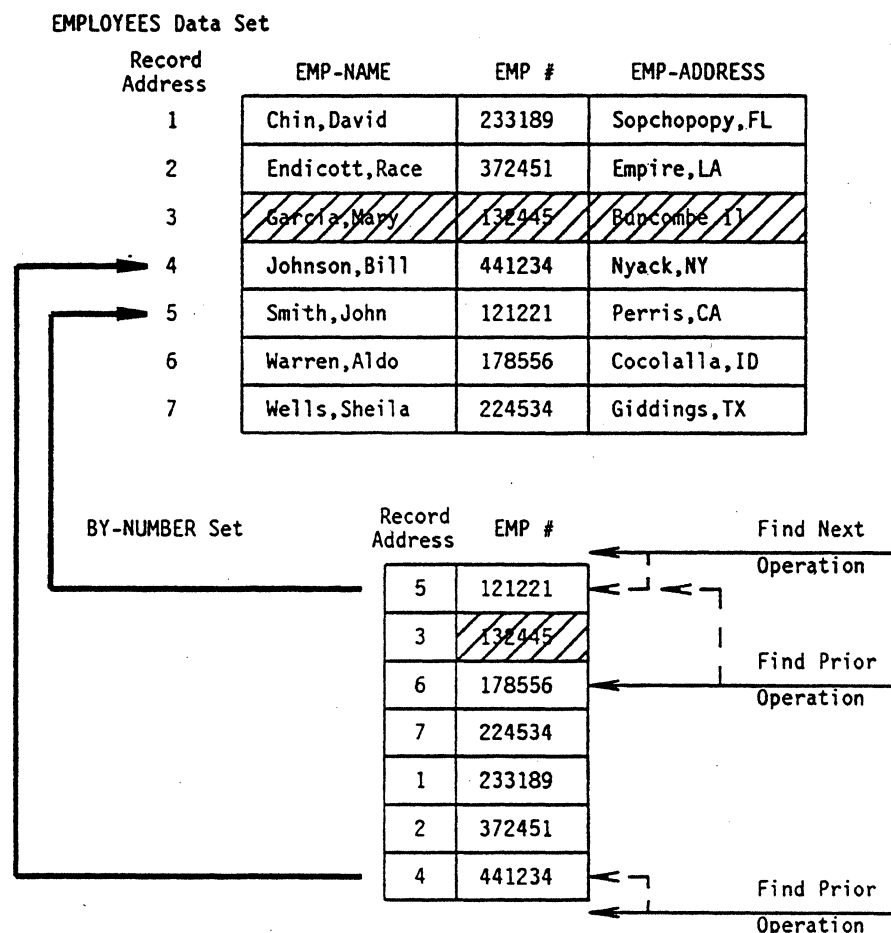


Figure 5-2. Resetting the Current Path for a Set

Example 1: Find Next

If the current path initially points to the beginning of the set, a find next operation on the set moves the current path to the first entry in the set (EMP #121221). Physical record 5 in the data set (for John Smith) is retrieved.

Example 2: Find Prior

If the current path initially points to the end of the set, a find prior operation on the set moves the current path to the last entry in the set (EMP #441234). Physical record 4 in the data set (for Bill Johnson) is retrieved.

Example 3: Find Prior with a Deleted Record

If the current path initially points to the third record in the set (EMP #178556) and the record for Mary Garcia is deleted in the data set, a find prior operation on the set moves

the current path to the first record in the set (EMP #121221). Physical record 5 in the data set (for John Smith) would be retrieved.

Adjusting the Path for Deleted Records

Whenever a record is deleted, all program paths pointing to the deleted record are marked as invalid. A find current operation returns a NOTFOUND exception.

Note: If a key item is changed in a lock and store operation, the operation is considered to be a delete operation followed by a create and store operation as far as all paths are concerned. This includes the set paths of the program performing the lock and store operation. All path adjustment rules apply.

Section 6

Retrieving Records through Search Operations

The DMSII data management operations find, lock, modify, secure, and delete are search and retrieval operations. These operations locate a specific data set record and place a copy of the record in your user work area for the data set. Your program can then access the copy of the record, and the information contained in the record. Table 6-1 briefly describes each of the operations.

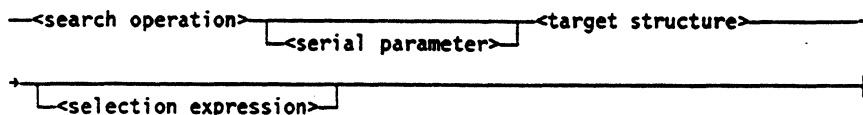
Table 6-1. Search and Retrieval Operations

Operation	Explanation
Delete	Locates a record, locks it, and deletes it. The current path becomes invalid; it points to a virtual record.
Find	Locates a record. The current path points to the retrieved record.
Lock	Locates a record and locks the record so that no other user can access it. The current path points to the retrieved record. A lock operation is required when you want to update a record.
Modify	Functions the same as the lock operation.
Secure	Locates a record and secures the record so that other users also can read, find, or secure the record. The current path points to the retrieved record. Other users cannot lock the record. A secure operation is useful when you want to retrieve data and guarantee that the record does not change during the course of a transaction.

Coding a Search Operation

All search operations have the same basic syntax. Although the text examples in this section show variations of the find operation, you can substitute any of the search operations.

<search operation>



The variable elements of the search operation are explained below:

Element	Explanation
<search operation>	Any of the search and retrieval operations: find, delete, lock, modify, or secure.
<serial parameter>	A serial search parameter: first, next, prior, or last.
<target structure>	The name of a database, data set, set, or subset. If the target structure is a database, DMSII retrieves the record containing the global data items for the database. If the target structure is a data set, set, or subset, DMSII retrieves a data set record. Data set records can be retrieved directly or through a set or a subset. Sets and subsets are used as pointers to the data set records.
<selection expression>	A criteria for selecting a record.

Understanding What Happens to a Retrieved Record

If the record is found, a copy of the record is placed in your user work area for that data set. Each invocation of the data set has its own copy of the record—the actual record remains in the database.

Multiple users can access the data concurrently without affecting the actual record. The actual record is changed only by two operations:

- The delete operation deletes the record.
- The store operation replaces an existing record or adds a new record.

Identifying the Types of Searches

A search can be serial, random, or a combination of both serial and random.

- A serial search is also known as a linear search. The search is based on the physical location of records and uses direction (toward the beginning or end of the structure) to locate the record. The direction is specified by a serial parameter: first, next, last, and prior.
- A random search is based on values in the records. The values are specified in a selection expression, such as locate all records where an employee has dependent children. DMSII uses binary search techniques, partial binary search techniques, or hash algorithms to locate the record.
- A combined search uses both the physical location and values to locate the record.

Serial search techniques are explained under “Searching Serially by Physical Location.” Binary techniques are outlined under “Using Binary Search Techniques for Sets and Subsets.” Partial Binary techniques are described under “Using Partial Binary Searches for Duplicate Keys.”

Searching Serially by Physical Location

A serial search retrieves a record by using its physical position in a structure. The structure can be a data set, set, or subset. The search starts at the current record and moves in a linear fashion toward the first or last record in the structure. The search can include one of the serial parameters: first, next, last, or prior.

As the number of records to be searched increases, the effectiveness of a serial search decreases. When your program needs to search through many records, use a combined search rather than a serial search.

Whenever a serial search is performed, whether by itself or in a combined search, the compiler prints a linear search warning in the compile listing to make you aware of the overhead involved in a serial search.

Note: Any find operation on a data set always forces a serial search.

Examining Each Record in a Serial Search

A serial search examines each item in turn, starting with the current record and proceeding sequentially. Each record is examined; none are skipped. The search can move from the current record either to the end of the structure (a forward serial search) or to the beginning of the structure (a backward serial search).

Using a Forward Serial Search

A forward serial search moves from the current record to the end of the structure. Figure 6-1 shows a forward serial search that starts from the beginning of the structure.

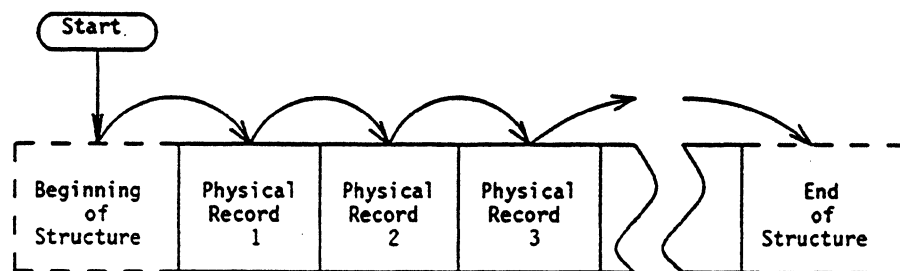


Figure 6-1. Forward Serial Search

If the serial search goes toward the end of the structure, it stops when it reaches the target record or, if the record is not found, the physical end of the structure. If the target record is not between the current record and the last record, inclusively, a NOTFOUND exception is returned.

If you are serially searching the entire structure, use the set to beginning operation to set the current path to the beginning of the structure before issuing a forward serial

Retrieving Records through Search Operations

search (such as a find next). Read “Using a Set Operation” in Section 5 for more information on the set operation.

NOTFOUND exceptions are detailed in Appendix A, “DMSII Exceptions and Errors.”

Using a Backward Serial Search

A backward serial search moves from the current record to the beginning of the structure. Figure 6-2 illustrates a backward serial search that starts from the middle of the structure.

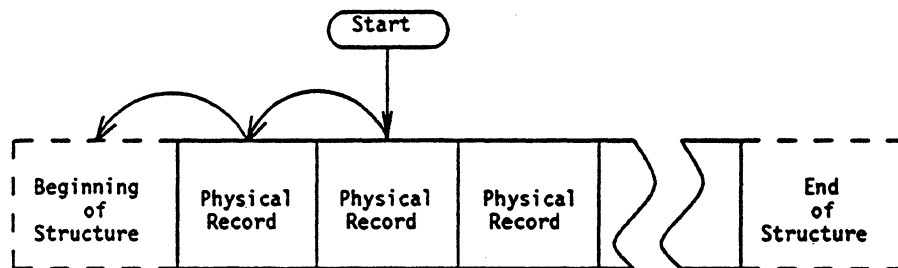


Figure 6-2. Backward Serial Search

If the serial search goes toward the beginning of the structure, it stops when it locates the target record or, if the record is not found, when it reaches the physical beginning of the structure. If the target record is not between the current record and the first record, inclusively, a NOTFOUND exception is returned.

If you are serially searching the entire structure, use the set to ending operation to set the current path to the end of the structure before issuing a backward serial search (such as a find prior). Read “Using a Set Operation” in Section 5 for more information on the set operation.

NOTFOUND exceptions are detailed in Appendix A, “DMSII Exceptions and Errors.”

Using the Serial Parameters

A search operation can specify one of the following serial parameters: first, next, prior, or last. These parameters give the direction of the serial search and request DMSII to retrieve the appropriate physical record. If no parameter is specified, DMSII retrieves the current record. Table 6-2 summarizes the effects of using the parameters.

Table 6-2. Serial Search Parameters

Parameter	Explanation
First	Retrieves the first physical record in the structure.
Next	Retrieves the next physical record (the current record plus one) in the structure. This parameter is usually preceded by a set to beginning operation.
Prior	Retrieves the prior physical record (the current record minus one) in the structure. This parameter is usually preceded by a set to end operation.
Last	Retrieves the last physical record in the structure.

If you are using the language extensions and you do not specify a serial parameter (first, last, next, or prior) – only a selection expression – DMSII performs a find first operation. For example, the following two delete statements are treated the same:

```
DELETE SETNAME WHERE KEY=12345
DELETE FIRST SETNAME WHERE KEY=12345
```

Figures 6-3 through 6-6 illustrate the records that are retrieved when you specify a particular parameter.

Retrieving the First Physical Record

As shown in Figure 6-3, a find first operation retrieves the first physical record in the structure.

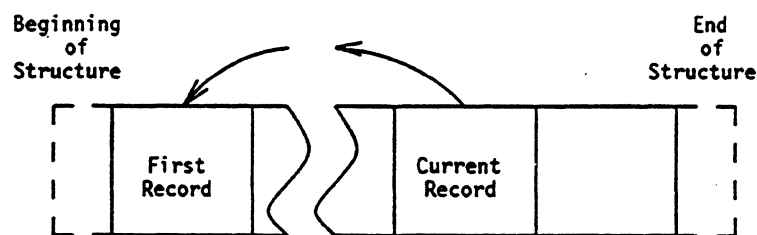


Figure 6-3. Retrieving the First Physical Record

Retrieving the Next Physical Record

As shown in Figure 6-4, a find next operation retrieves the next physical record (the current record plus one) in the structure.

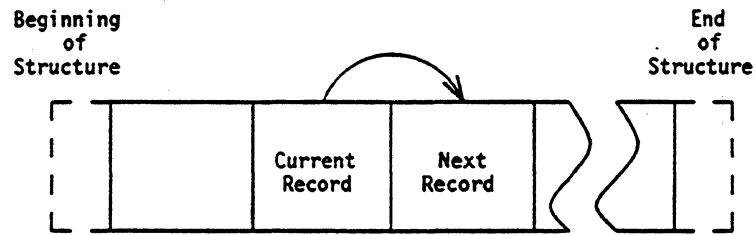


Figure 6-4. Retrieving the Next Physical Record

Retrieving the Prior Physical Record

As shown in Figure 6-5, a find prior operation retrieves the previous physical record (the current record minus one) in the structure.

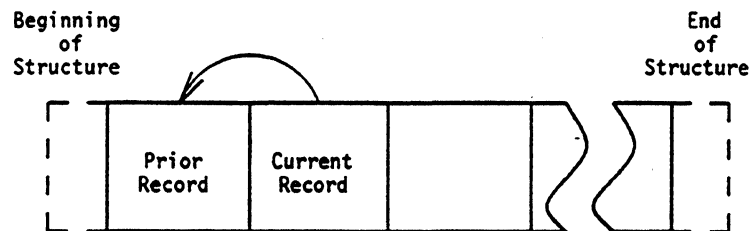


Figure 6-5. Retrieving the Prior Physical Record

Retrieving the Last Physical Record

As shown in Figure 6-6, a find last operation retrieves the last physical record in the structure.

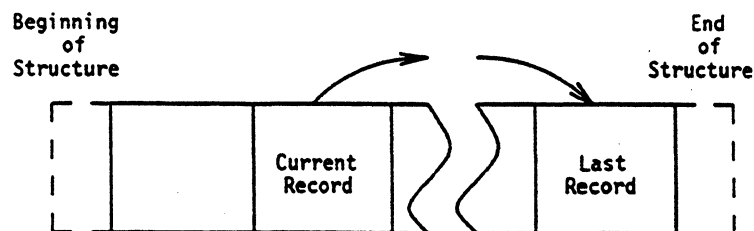


Figure 6-6. Retrieving the Last Physical Record

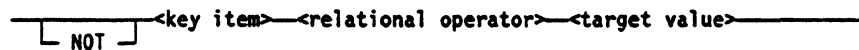
Searching Randomly by Selection Expression

A *random search* retrieves a record by searching a target structure for values stored in the record. The values, known as *keys* or *key items*, are specified in a Boolean selection expression. A random search for a key item can use either binary search techniques or a hashing algorithm. (See “Using Binary Search Techniques for Sets and Subsets” later in this section for information on binary search techniques.)

Defining the Components of Selection Expressions

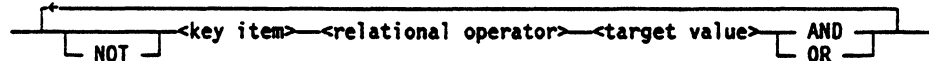
A selection expression consists of three parameters: a key item, a relational operator, and a target value. Optionally, the selection expression can be expressed as a negative condition by using the Boolean NOT operator.

<selection expression>



One or more criterion can be strung together in the same selection expression using the Boolean operators AND and OR. This is known as a complex selection expression.

<complex selection expression>



The variable elements of the selection expression are explained below:

Element	Explanation
NOT	Changes the selection criteria to the negative (reverse) of the condition.
<key item>	Identifies the data item used to sort the file. The value in the key item is compared to the target value.
<relational operator>	Identifies one of the following arithmetic comparison operators: =, >, <, =>, =<, <>.
<target value>	Identifies the value that is compared to the value in the key item. The type of the value must be the same as the type of the key item value.
AND	Enables you to combine conditions so that both conditions must be true.
OR	Enables you to combine conditions so that only one of the conditions must be true.

The way in which you code your selection expression determines whether the search operation is binary, partially binary, or serial. Your code also determines if the search is for an exact match or a relative value. Examples of different ways of coding a selection expression can be found in this section.

As records are examined, the selection expression evaluates to either TRUE or FALSE. The record being examined either meets the criteria (TRUE) or does not (FALSE). If it

Retrieving Records through Search Operations

evaluates to TRUE, the search stops. If it does not, the search continues until a record that meets the criteria is found or a NOTFOUND exception is returned.

NOTFOUND exceptions are detailed in Appendix A, "DMSII Exceptions and Errors."

Notes:

- *If your application programming language supports language extensions, the syntax and semantics of the selection expression are detailed in the appropriate programming language reference manual.*
- *You can use a link item anywhere a selection expression is used.*

Selecting a Target Structure

Your choice of a target structure is limited by your application program interface.

- For the interpretive interface, the target structure can be a data set, a set, or a subset. See "Understanding Search Operation Differences for the Interpretive Interface" later in this section for an explanation of how the interpretive interface handles selection expressions in data set retrievals.
- For the language extensions, the target structure of a serial search can be a data set, a set, or a subset. However, if you use a selection expression to retrieve a record, the target structure cannot be a data set — the record must be retrieved through a set or subset.

When the target structure is a set or subset, a random search operation retrieves the first record that matches the selection criteria.

Selecting Key Items

A *key item* is a data item defined in the DASDL that can be used to sort the data set. By definition, a key item is a required item. In search operations, the value stored in the key item is compared to the target value.

A key item can be simple or complex. A simple key item is composed of a single data item. For example, the following key item, LEVEL, is a simple key item:

```
MANAGERS SET OF EMPLOYEES  
KEY IS LEVEL;
```

A complex key item is composed of two or more data items. The first defined item is known as the major key. All other items are minor keys. In the following example, LEVEL is the major key. EMPNO and PROJ are minor keys:

```
MANAGERS SET OF EMPLOYEES  
KEY IS (LEVEL, EMPNO, PROJ);
```

Consult the DMSII Technical Overview and the DMSII DASDL Reference Manual for a detailed explanation of key items.

Note: The find key of operation can also use key data items. See "Find Key Of Operations for Sets and Subsets" later in this section.

Using Complex Key Items

In your selection expression, you can enter the various key items that compose a complex key in any order. However, because DMSII always constructs the complex key item in the order it is defined in the DASDL source file, the search is faster and more efficient if you enter the major and minor key items exactly as they are defined by DASDL.

For the fastest performance, your selection expression should contain the major key item. Only include a minor key if it is needed to uniquely identify the record. For example, if the complex key is defined as (LEVEL, EMPNO, PROJ):

- Always use LEVEL in the selection expression.
- If the value of LEVEL is not enough to uniquely identify the record, also use the minor key EMPNO.
- If the value of LEVEL and EMPNO are not enough to uniquely identify the record, add the minor key PROJ.

You can skip minor key items. You could use LEVEL and PROJ and not use EMPNO. But, because of the way DMSII constructs the complex key item, skipping minor keys slows down processing time.

Using Key Item Order

Key items in index sequential and ordered list sets and subsets can be stored and accessed in either ascending or descending order. When ascending order is used, key values go from lowest to highest. When descending order is used, key values go from highest to lowest. By default, ascending order is used.

If the key item has been defined in the DASDL as having an ascending or a descending order, the selection expression can direct the search by using relational operators in conjunction with the first, next, prior, and last search parameters.

Read "Finding an Approximate Match" later in this section for more information on using ascending or descending key order in search operations.

Using Duplicate Key Items

The duplicate option in DASDL allows index random, index sequential, and ordered list set and subsets to have two or more records with identical key values. By default, no duplicates are allowed—each key value must be unique.

If duplicate keys are allowed, use a combined search operation (with a serial parameter and a random selection expression) to ensure that you locate the correct record.

Retrieving Records through Search Operations

Read "Combining Serial and Random Searches" and "Using Partial Binary Searches for Duplicate Keys" later in this section for more information on duplicate keys.

Using the Same Key Item More Than Once

When criterion are strung together in one selection expression, the same key item can appear in more than one condition in the same search operation. However, if the same key item is needed, it is more efficient to have separate search operations.

For example, if you are searching for managers whose levels are above 14 and less than 5, you could use the same key item as shown below:

```
FIND MANAGER WHERE LEVEL>14 AND LEVEL<5
```

Or, you could you two search operations.

```
FIND MANAGER WHERE LEVEL>14  
FIND MANAGER WHERE LEVEL<5
```

The series of searches would be more efficient.

Read "Using the AND Operator to Test for More than One Criterion" and "Using the OR Operator to Test for More than One Criterion" for more information on efficient coding of multiple key items.

Using Nonkey Items in Selection Expressions

You can use nonkey items in selection expressions. However, using a nonkey item usually forces DMSII to perform a serial search.

If an item is not required, it can be tested to see if it has a value of null.

Testing for Relationships

A *relational operator* specifies a relationship. DMSII compares the value in the key item to the value in the target item to see if the specified relationship is true or false.

The basic operations, their most common arithmetic symbols, and their most common mnemonics are shown in Table 6-3. Consult the appropriate programming language reference manual for the valid symbols and mnemonics in a particular language.

Table 6–3. Relational Operators in a Selection Expression

Relational Operator	Arithmetic Symbol	Common Mnemonic
Equal to	=	EQL
Greater than	>	GTR
Less than	<	LSS
Equal to or greater than	=>	GEQ
Equal to or less than	=<	LEQ
Not equal to	<>	NEQ

The operators are computed from left to right. To alter how they are evaluated, use parentheses.

Examples of Using Relational Operators

In the following examples, the value of the key item PROJ is being tested to see if the relationship is true or false.

Example	Meaning
FIND PROJ = CURRENT	If the value of PROJ is equal to the value of CURRENT, the relationship is true.
FIND PROJ GTR 35	If the value of PROJ is greater than 35, the relationship is true.

Selecting a Target Value

The target value is compared to the value in the key item. The target value can be either a constant, a variable, or a combined expression, as defined in the application language. In addition, COBOL programs can use the value in a data item as the target value.

The type of the value must be the same as the type in the key item. For example, if the key item is a numeric type, then the target must also be a numeric type.

Finding Exact Matches

Use the EQL operator to search for an exact match. For alpha items, the key item and target value should have the same number of characters.

Matching Alphanumeric Items Exactly

For alphanumeric items, the value in the key item is examined from left to right. If the target value has less characters, blanks () are added to the right of the target to form a new target value. This is known as left justification.

For example, if the alphanumeric key item is defined as having five characters and the target value has only three, two blanks are added to the target value; a target value *a2c* becomes *a2c* .

The target value is truncated when it exceeds the number of characters in the key item; a target value of *abcd124fg* would be truncated to *abcd1* if the key item is defined as having five characters.

Matching Numeric Items Exactly

For numeric items, the value in the key item is examined algebraically.

If the target has more digits than the key item, the target is truncated. For example, if the key item has five digits and the target has seven digits, the rightmost digits in the target are truncated. A target of *1234567* becomes *12345*.

If a number does not have as many digits as were defined, zeroes are added to pad the number. For example, if a number is defined as three digits, the number *2* becomes *002*.

If the key item is defined as signed and the target is signed, the sign is part of the search. If the key item is signed and the target is unsigned, the target is assumed to be a positive number. For unsigned key items, any sign in the target is ignored.

Finding an Approximate Match

Use the LSS, GTR, GEQ, and LEQ operators to search for an approximate match. The key item must be arranged in either ascending or descending order.

Key values are evaluated using the EBCDIC collating sequence. Depending on the search operation and the order of the key item (ascending or descending), some methods use the EBCDIC collating sequence more efficiently.

Table 6-4 gives the most efficient combinations for ascending keys.

Table 6—4. Efficient Combinations for Search Operations Using Ascending Keys

Operation	Relational Operator
Find first or find next	GTR (>)
Find prior or find last	LSS (<)

Table 6-5 gives the most efficient combinations for descending keys.

Table 6-5. Efficient Combinations for Search Operations Using Descending Keys

Operation	Relational Operator
Find first or find next	LSS (<)
Find prior or find last	GTR (>)

Note: The combinations shown in Tables 6-3 and 6-4 force a partial binary search. If you have duplicate keys, read "Using Partial Binary Searches for Duplicate Keys" later in this section.

Using Boolean Operators

You can use the AND, OR, and NOT Boolean operators in a selection expression to determine if a relationship is TRUE or FALSE.

Table 6-6 is a truth table which shows how to evaluate a selection expression. For example, if the relational operator A is TRUE and the relational operator B is also TRUE, the table shows that:

- The Boolean statements A AND B and A OR B evaluate to TRUE.
- All Boolean NOT A, NOT B, NOT (A AND B), and NOT (A OR B) statements evaluate to FALSE.

Table 6-6. Truth Table for Boolean Operators

A	B	A AND B	A OR B	NOT A	NOT B	NOT (A AND B)	NOT (A OR B)
True	True	True	True	False	False	False	False
True	False	False	True	False	True	False	True
False	True	False	True	True	False	False	True
False	False	False	False	True	True	True	True

All Boolean operators take precedence over relational operators.

Using the NOT Operator

A selection expression can be expressed as a negative condition by using a NOT operator. The NOT operator appears before the selection expression. In the following example, the set MANAGER is searched for the first record in which the manager level is not 13.

FIND MANAGER AT LEVEL NOT =13

Retrieving Records through Search Operations

Be careful when using negative selection criteria to eliminate all unwanted conditions. The preceding example could return a record in which the level is 12 or 100. For more efficient and precise searches, it is better to specify what you want rather than what you do not want.

Using the AND Operator to Test for More than One Criterion

By using the Boolean operator AND, you can use more than one criterion in the same search operation. The target structure remains the same, but a string of key items, relational operators, and target values are connected by the keyword AND.

An AND operator makes the selection criteria additive. Each criterion in the string must be TRUE. For example, the following binary search looks for a manager who is a level 13 and has five employees. If the manager is only a level 13 or only has five employees – not both – there is no match.

```
FIND MANAGER AT LEVEL=13 AND NOEMP=5
```

To code a complex AND selection expression so that it produces the most efficient binary search, use the following guidelines:

- Specify needed key items in the same order they are defined in DASDL. For example, if the first key item is LEVEL, the second key item is EMPNO, and the third key item is PROJ, then always specify LEVEL before EMPNO and EMPNO before PROJ.
- Do not use any insignificant key items. If LEVEL is enough to uniquely identify the record, do not use any minor key item. If LEVEL and EMPNO are enough, do not use PROJ.
- Do not skip key items. Do not use LEVEL and then PROJ, skipping EMPNO. Do not use EMPNO and then PROJ, skipping LEVEL.
- Use each key item only once.
- Ensure that the relational operator for all but the last AND selection expression is the equal to (=) operator.
- Link all AND conditions. An OR condition cannot appear between AND conditions.

If your selection expression does not follow these guidelines, the resulting search might not be a binary search or the search process can be impaired. For example, if you skip a minor key, DMSII searches for and inserts the skipped key, slowing down processing time.

Examples of Using AND Operators

Following are two sets of examples. Because of the coding of the selection expressions, the first set produces binary searches. The second set produces partial binary or serial searches.

All examples use the set MANAGER with three key items: LEVEL, EMPNO, and PROJ.

Example 1: Binary Searches

In the first example, each key item is specified as it was defined in DASDL. No key item appears more than once. No key item is skipped.

```
FIND MANAGER AT LEVEL=13 AND EMPNO=7 AND PROJ=CHART
```

In the second example, only the last key item using the AND operator has a relational operator other than equal to (=). Since PROJ was not needed for this search, it is not part of the criterion in the selection expression.

```
FIND MANAGER AT LEVEL=13 AND EMPNO=>6
```

Example 2: Serial Searches

In the first example, the first key is skipped, forcing DMSII to perform a serial search.

```
FIND MANAGER AT EMPNO=7 AND PROJ=CHART
```

In the second example, key items are not sequenced as they were defined in the DASDL. This initiates a serial search.

```
FIND MANAGER AT LEVEL=13 AND PROJ=CHART AND EMPNO=>6
```

The next example forces a serial search by using the less than (<) operator in the first selection expression.

```
FIND MANAGER AT LEVEL<13 AND EMPNO=>6
```

In the last example, the OR operator is interspersed with the AND operators, forcing a serial search.

```
FIND MANAGER AT LEVEL=13 OR EMPNO=>6 AND PROJ=CHART
```

Using the OR Operator to Test for More than One Criterion

By using the Boolean operator OR, you can use more than one criterion in the same search operation. The target structure remains the same, but a string of key items, relational operators, and target values are connected by the OR operator.

The OR operator forces a serial search. If the search operation only uses the OR operator, the entire search is serial. If the search operation combines AND and OR operators, the OR operator forces a serial search of the OR conditions; the AND operator can cause either a binary or serial search.

The OR operator allows choices. Only one of the specified relations must be TRUE. For example, the following search looks for a manager who has more than 200 employees or whose project is CHART. If either condition is TRUE, there is a match.

```
FIND MANAGER AT EMPNO>200 OR PROJ=CHART
```

Retrieving Records through Search Operations

Use the following guidelines when constructing a search with the OR operator:

- The key items can be given in any order.
- The same key item can be used in a string of OR operators.
- Any relational operator can be used in each OR selection expression.

Examples of Using OR Operators

The following examples use the OR operator. All examples use the data set **MANAGER** with three key items: **LEVEL**, **EMPNO**, and **PROJ**.

The first example shows an invalid selection expression. Each selection expression must have a relational operator. There is no relational operator between **PROJ** and **CHART**.

```
FIND MANAGER AT LEVEL>13 OR PROJ CHART
```

The next example shows that the key items do not have to be specified in the same order they are defined in the **DASDL**.

```
FIND MANAGER AT PROJ=CHART OR EMPNO=3
```

Using AND, OR, and NOT Operators Together

When the Boolean **AND**, **OR**, and **NOT** operators appear in the same search operation, the following rules apply:

- Conditions enclosed in parentheses are evaluated first.
- **NOT** operations are performed second.
- **AND** operations are performed third.
- **OR** operations are performed last.

The **NOT** operator is only active for the selection expression immediately following the operator; it does not apply to any successive expression. This means that the operation

```
FIND MANAGER AT LEVEL NOT=13 OR LEVEL<13
```

is parsed as

```
FIND MANAGER AT (LEVEL NOT=13) OR (LEVEL<13).
```

Examples of Using AND, OR, and NOT Operators

All the following examples search the set **MANAGER** for managers who are level 14 or above. Only the key item **LEVEL** is used. The last example demonstrates the most efficient code.

```

FIND MANAGER AT LEVEL NOT=13 OR LEVEL NOT<13
FIND MANAGER AT LEVEL>13 OR LEVEL NOT=13
FIND MANAGER AT LEVEL NOT=13 AND LEVEL NOT<13
FIND MANAGER AT LEVEL=14 OR LEVEL=>15
FIND MANAGER AT LEVEL NOT=<13
FIND MANAGER AT LEVEL=>14
    
```

The next examples search the set **MANAGER** for managers who are level 14 or above and have more than five employees working on the project **CHART**. The key items have been defined in the DASDL in the following order: **LEVEL**, **NOEMP**, and **PROJ**. The last example demonstrates the most efficient code; however, each example generates a serial search.

```

FIND MANAGER AT LEVEL NOT=13 AND LEVEL NOT<13 AND NOEMP>5 AND PROJ=CHART
FIND MANAGER AT LEVEL=14 OR LEVEL=>15 AND NOEMP>5 AND PROJ=CHART
FIND MANAGER AT LEVEL=>14 AND NOEMP>5 AND PROJ=CHART
    
```

Combining Serial and Random Searches

A serial search uses one of the serial parameters (first, next, last, or prior) to locate a record. A random search uses a selection expression to locate a record. A combined search uses both a serial parameter and a selection expression to locate a record. Use a combined search when more than one record in a target structure could meet the given selection criteria.

Table 6-7 shows the various combinations of serial and random searches. In all cases, if the record is found, it becomes the current record.

Table 6-7. Combined Search and Retrieval Operations

Operation	Explanation
find first <target structure> <selection expression>	The current path is set to the beginning of the structure. The structure is searched for the first physical record that meets the selection criteria.
find last <target structure> <selection expression>	The current path is set to the end of the structure. The structure is searched backward (toward the beginning) for the first physical record that meets the selection criteria.
find next <target structure> <selection expression>	The search starts at the current record and moves toward the end of the structure. The first physical record that meets the selection criteria is retrieved.
find prior <target structure> <selection expression>	The search starts at the current record and moves toward the beginning of the structure. The first physical record that meets the selection criteria is retrieved.

Find Key Of Operations for Sets and Subsets

By using a special form of the find operation, *find key of*, you can retrieve data in a set or subset. A find key of operation is usually used to verify that a record exists in a data set or if you want to reject/accept records without performing an input/output operation against the data set.

Key data items are data items that are stored in both the data set and the set but are not key items. The DASDL for the database must explicitly declare the key data items in both the data set and the set or subset.

When you use a find key of operation, the current path for the data set is not changed but the current path for the set or subset is changed. You should synchronize the current paths before executing your next search operation. Refer to Section 5, "Setting the Current Path."

A find key of operation must have a selection expression. The selection expression must contain items that are in the set or subset; it cannot reference items that are in the data set only. The items can be key items for the set or subset, or any key data items.

In the following DASDL example, the key item is EMP-NAME. The key data item is EMP-NBR. Both the key item and the data item can be used in a find key of operation.

```
EMP-NAME-LIST SET OF EMPLOYEE
KEY IS EMP-NAME
DATA EMP-NBR
```

However, because a key item determines how the set or subset is organized, the key item is a more efficient selection condition than a key data item. If the selection expression contains only key data—no key item—then only a serial search can be performed. If the selection expression contains a key item, then a binary or partial binary search can be performed.

Note: A find key of operation moves only the key items and key data items to the user work area. A find key of operation does not read the corresponding data set record. If you want to retrieve the corresponding data set record, use a find, lock, secure, modify, or delete operation.

Using Binary Search Techniques for Sets and Subsets

Binary searches are the fastest and most efficient searches. However, these searches are valid only for sets or subsets that have ordered keys. They cannot be used if duplicate keys are allowed. Read "Using Partial Binary Searches for Duplicate Keys" later in this section for more information on duplicate key searches.

A *binary search* divides the records in a set or subset into two halves, and by comparing the value of the selection criteria to the value stored in the midpoint record, the search keeps rejecting one half of the structure until the target is found. (Fractions are rounded up.) The corresponding data set record is then retrieved.

Example of Using Binary Search Techniques

Figure 6-7 shows an index sequential set in which the the key item is a numeric value stored in ascending order. The key items are the numbers 1 through 10.

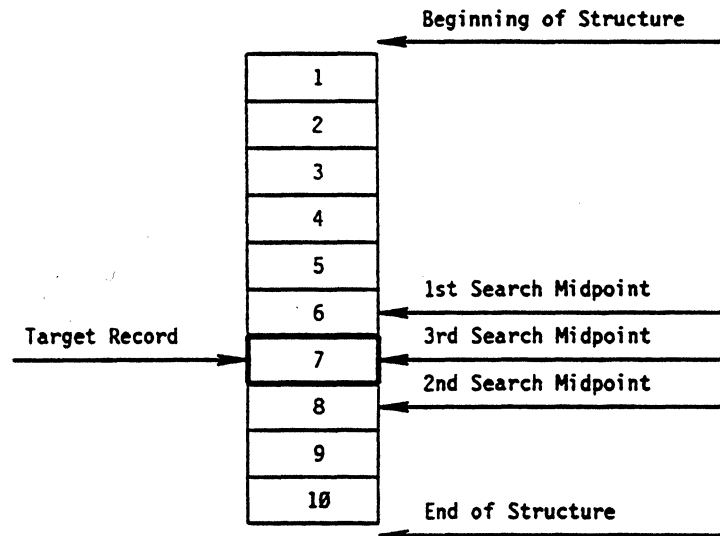


Figure 6-7. Binary Searches

If the selection criterion is the number 7, a binary search performs the following steps:

1. Calculates the total number of records (10 records).
2. Divides the total by 2 (5 records in each part).
3. Goes to the midpoint record (rounded up to record 6) and decides the following:
 - Is the value in the midpoint record equal to 7? If it is, the target has been found.
 - Is the value in the midpoint record less than 7? If it is, calculate the total number of records in the first part, divide by 2, and go to the midpoint.
 - Is the value in the midpoint record greater than 7? If it is, calculate the total number of records in the second part, divide by 2, and go to the midpoint.

In this example, the value in the midpoint record is 6. The binary search narrows down the scope of the search to records 6 through 10.

4. Calculates the total number of records to be searched (5 records).
5. Divides the total by 2 (2.5 records are rounded up to 3 records).
6. Goes to the midpoint record and decides the following:
 - Is the value in the midpoint record equal to 7? If it is, the target has been found.
 - Is the value in the midpoint record less than 7? If it is, calculate the total number of records in the first part, divide by 2, and go to the midpoint.
 - Is the value in the midpoint record greater than 7? If it is, calculate the total number of records in the second part, divide by 2, and go to the midpoint.

In this example, the value in the second midpoint record is 8; it is greater than 7. The binary search continues with records 6 through 8.

7. Calculates the total number of records to be searched (3 records)
8. Divides the total by 2.
9. Goes to the midpoint record (record 7) and compares the value to the target value.

The value in record 7 is 7; the same as the target value. The search is ended. DMSII then uses the value to locate and retrieve the corresponding record in the data set.

Using Partial Binary Searches for Duplicate Keys

DMSII does not retrieve a record by using only binary search techniques if key items in the data set can have the same value; that is, the DASDL definition of the data set allows duplicate keys. Instead, DMSII performs a partial binary search. For example, if a key item is the name of a project and seven people are assigned to the same project, DMSII uses a partial binary search.

A *partial binary search* uses a binary search to locate a record in the set or subset that meets the specified criteria (given in a selection expression) and then uses a serial search to locate duplicates. The ordering of the key items and the duplicates affects the direction of both the binary and serial searches. The most efficient run-time searches use at least a partial binary search.

One of the following conditions must be met in the selection expression to force a partial binary search:

- The key items specified include the highest-order key item with no key items omitted between the most and least significant key items specified.
- All key conditions on individual key items are connected by the operator AND.
- For key items arranged in ascending order, only the highest-order key item uses the operators EQL, GTR, or GEQ.
- For key items arranged in descending order, only the highest-order key item uses the operators EQL, LSS, LEQ.

Figure 6-8 shows the combination of find operations, key ordering, and operators that initiate a partial binary search.

Serial Search Parameter	Key Order	Relational Operator
First	Ascending	= > =>
	Descending	= < =<
Last	Ascending	= > =<
	Descending	= < =>
Next	Ascending	= > =>
	Descending	= < =<
Prior	Ascending	= < =<
	Descending	= > =>

LEGEND

- = Equal to
- > Greater than
- => Equal to or greater than
- < Less than
- =< Equal to or less than

Figure 6-8. Find Operations That Force a Partial Binary Search

Any of the following in the coding of a selection expression can force a long search:

- Using an OR operator
- Omitting the major key item
- Using an arithmetic operator improperly

If DMSII performs a partial binary search, the compiler prints a serial search warning in the compile listing.

Note: *If the selection expression uses the highest order key with an EQL operator, the search continues only until it finds the record. (This is known as early termination.) Otherwise, the search continues until the end of the set is reached.*

Consult the DMSII Technical Overview and the DMSII DASDL Reference Manual for more information on key items.

Optimizing a Search with Reblocking and Readahead

Reblocking enables you to optimize both random and serial searches by maintaining two simultaneous block sizes for the same structure. A small block size is used for random searches, and a large block size is used for serial searches. Reblocking is initiated using a find, secure, lock, modify, or delete operation and a current, first, prior, next, or last serial parameter.

When reblocking is in effect, the Access routines are instructed to perform readahead for find, lock, modify, delete, or secure operations for serial searches. *Readahead* is reads of the database in which the last read operation can be used to reduce the time the program must wait for database I/O operations to complete.

The DASDL options REBLOCK, REBLOCKFACTOR, and BLOCKSIZE define whether reblocking is allowed and determine the degree of reblocking. Consult the DMSII DASDL Reference Manual for specific information about the options.

Examples of Locating Data Set Records

The following DASDL definition and COBOL program fragment are used to illustrate how data set records are located by different find operations.

DASDL Definition

This DASDL code defines one data set and one set. The data set record contains one numeric item. The set uses the numeric item as its key.

```
D DATA SET
(
  N      NUMBER (9)
);
S SET OF D KEY N;
```

COBOL Program Fragment

This COBOL program fragment invokes the database and specifies which records to retrieve.

DATA-BASE SECTION.

DB EXAMPLEDB.

Ø1 D.

Ø1 S1 = S.

Ø1 S2 = S.

Ø1 S3 = S.

*

FIND S1 AT N=3.

FIND S2 AT N=10.

FIND S3 AT N=9.

FIND FIRST D.

Results of Find Operations

Figure 6-9 depicts the content of an index sequential set and a standard data set.

The data set records are not arranged in any order. The set records are arranged in ascending key order, from 1 to 12. The variables D, S, S1, S2, S3 and N are defined in the program.

The program has initially positioned three current paths in the set and one current path for the data set.

- The initial current path for S1 points to the set record where N is 3 and retrieves the data set record where N is 3.
- The initial current path for S2 points to the set record where N is 10 and retrieves the data set record where N is 10.
- The initial current path for S3 is invalid. There is no set record where N is 9. There is no data set record where N is 9. The current path points to a virtual record.

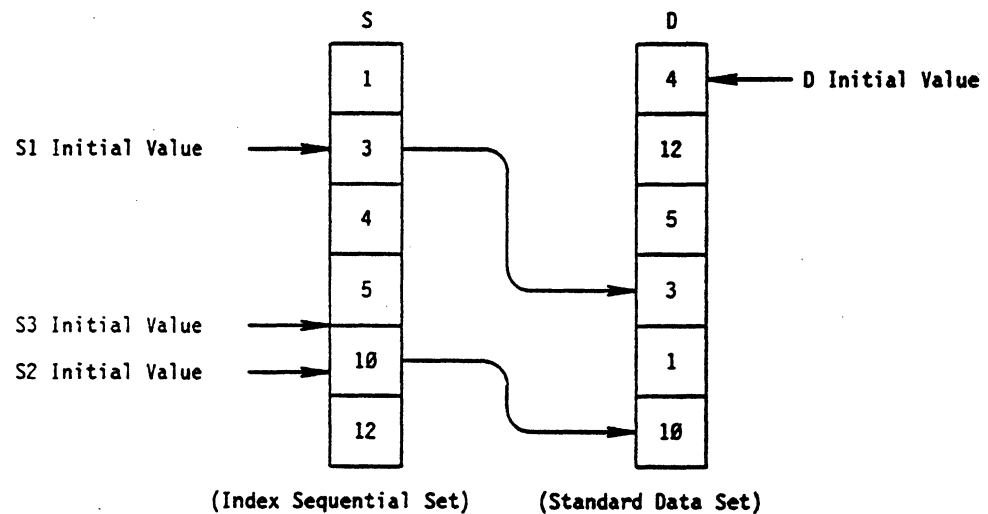


Figure 6-9. Locating Data Set Records through a Set

Example 1: Find (current) Operations

FIND S1 retrieves the data set record where N is 3.

FIND S2 retrieves the data set record where N is 10.

FIND S3 results in an exception.

FIND D retrieves the data set record where N is 4.

Example 2: Find Next Operations

FIND NEXT S1 moves the current path to the next physical record in the set, retrieving the data set record where N is 4.

FIND NEXT S2 moves the current path to the next physical record in the set, retrieving the data set record where N is 12.

FIND NEXT S3 moves the current path to the next physical record in the set, retrieving the data set record where N is 10.

In this example, when FIND NEXT S3 is completed, the current path for the data set points to the last record in the data set. A command to FIND NEXT D returns an exception; there is no next physical record in the data set.

Example 3: Find Prior Operations

FIND PRIOR S1 moves the current path to the prior physical record in the set, retrieving the data set record where N is 1.

FIND PRIOR S2 moves the current path to the prior physical record in the set, retrieving the data set record where N is 5.

FIND PRIOR S3 moves the current path to the prior physical record in the set, retrieving the data set record where N is 5.

FIND PRIOR D moves the current path to the prior physical record in the data set, retrieving the data set record where N is 12.

Understanding Search Operation Differences for the Interpretive Interface

The search operations of the DMINTERPRETER library perform the same functions as the DMSII search language extension but with more flexibility.

Evaluating Selection Expressions

The DMINTERPRETER library determines how to process selection expressions at run time. To reduce overhead, the most recent find operation is saved in an internal form,

and a selection expression is reevaluated only when it changes. As a result, successive identical calls on the DBFIND entry point involve less overhead.

Finding a Data Set Using Selection Expressions

The interpretive interface can use a data set name to locate a record that meets the conditions specified in a selection expression. When a data set is specified in a selection expression, the interpretive interface uses the set or Access that provides the most efficient search. Subsets are not considered. If no sets or Accesses are available, DMSII serially searches the data set records.

The benefits of this arrangement are as follows:

- Less knowledge of the database is required.
- Data independence is improved.
- If a new set is added that could be used more efficiently, existing programs do not need to be rewritten to take advantage of this set.

However, when the search condition changes, the interpretive interface decides if a different index set should be used. If a different index set is chosen, a call on the DBFIND entry point could produce unexpected results. For example, a find next operation might actually return a prior record. The interpretive interface returns an exception when such a situation is detected. This problem can be avoided by following any of these guidelines:

- Do not change the condition in the middle of a search on a data set.
- Specify an index set instead of a data set as a DBFIND parameter.
- Call either the DBSET or DBFIND entry point with the direction parameter of first or last before you change the condition parameter.

Terminating Serial Searches Early

In some cases, when the interpretive interface performs a serial search, it does not search the entire structure being used. For example, if the search specifies an LEQ parameter on an ascending key item, the search terminates the first time the search condition fails. This form of serial search can, at times, be faster than a binary search.

Note: When serial and binary searches are combined, the serial search is also subject to the same early termination when the search condition becomes false.

Performing a Binary Search on a Complex Key

To perform a binary search on a complex key, the DMSII language extensions require an equality test for all but the most minor key specified. Otherwise, a serial search of the index is performed.

Example 1

In the following example, the FIND language extension produces a serial search; the equivalent ALGOL interpretive interface coding produces a binary search.

Language Extension

```
FIND PART AT PARTNO >10 AND QUANTITY >20;
```

Interpretive Interface

```
STRING EMPTY;  
EMPTY:=" ";  
DBFIND(EMPTY,"PART","PARTNO>10 AND QUANTITY>20");
```

Example 2

Below is the DASDL source code for the next example. In the example, the interpretive interface performs a binary search on items inside a GROUPKEY.

```
DASDL:  GROUPDATA DATA SET  
        (GROUPKEY GROUP  
          (GROUPITEM NUMBER(2);  
           OTHERITEM ALPHA(10)));  
GROUPSET SET OF GROUPDATA KEY IS GROUPKEY;
```

The following FIND NEXT language extension produces a serial search of GROUPSET. The interpretive interface coding produces a binary search.

Language Extension

```
FIND NEXT GROUPSET AT GROUPITEM = 99;
```

Interpretive Interface

```
DBFIND("NEXT","GROUPSET","GROUPITEM=99");
```

Note: Using a minor key in a find operation results in a serial search. For example, if either search uses OTHERITEM, the search is serial.

Using an Automatic Find Key Of Operation

A data set record is not actually moved into the DMSII user work area as a result of all calls on the DBFIND entry point. Instead, if a set is used, only the set entry is read and the data set read operation is marked as *pending*.

The interpretive interface completes the read if a statement is issued that requires one of the following:

- Data that is not in the key area
- A correct path to the data

As a result, a call on the DBFIND entry point that references only key items or key data items behaves like a find key of language extension. DBGET operations (which move data from the the DMSII user work area to variables that have been declared by the programmer) that reference only key items or key data items do not cause the data set record to be read; the data set record might never need to be read.

Guidelines for Search and Retrieval Operations

By using the current path and record, the most efficient search type and technique, and the available DASDL options, you can reduce the time it takes to search for and retrieve the correct data. When constructing a search operation, keep in mind the following points:

- Only serial searches can be performed on a data set – no matter what search type or technique is specified by the operation.
- If possible, retrieve records in a data set by using a set or subset to point to records. (Read the DMSII Technical Overview for a detailed explanation of sets and subsets.)
- You cannot use a search operation on a bit vector set or subset.
- When processing all records in a data set, serial searches are faster than random searches.
- A serial search always starts from the current record and is one-directional, either forward or backward.
- For faster processing, do not put selection conditions in a find next or find prior operation. Instead, place the selection conditions in a find first or find last operation, and then execute a find next or find prior operation, as appropriate.
- Code selection expressions to generate binary or partial binary searches.
- A selection expression should limit the amount of sequential processing in a partial binary search to a small portion of the set.
- Specify key items in your selection expressions in the same order they are defined in the DASDL.
- In a combined search, DMSII always performs the random search before performing the serial search.
- A successful find operation using a set, subset, or access alters the current path for the set and data set.
- A successful find operation using a data set alters only the current path for the data set.
- Each invocation of a data set creates an associated user work area.

Section 7

Updating the Database

Updating the database in an application program can involve any of the following tasks:

- Retrieving records
- Adding new records to a data set
- Changing values in a record
- Deleting records from a data set
- Adding or deleting key entries in manual subsets

A new or changed record does not become part of the database until it is explicitly stored in the database.

Selecting the Appropriate Means of Retrieving the Record

DMSII allows more than one program to retrieve the same record concurrently. If all the programs that access the database open the database for inquiry only, there is no problem. However, if one or more programs open the database for update and try to modify the same record, the database can become corrupted and processing problems can arise.

To avoid problems, carefully select how your program opens the database and retrieves a record.

When the Database Is Open for Inquiry Only

If your program opens the database for inquiry only, use either a find or secure operation to retrieve records.

- If your program uses a find operation, the record is not locked; it can be concurrently accessed and updated.
- If your program uses a secure operation, the record can be concurrently accessed but cannot be changed or deleted.

When the Database Is Open for Updates

If your program opens the database for updates, use a lock, modify, or delete operation to retrieve records.

- Use a lock or modify operation when you want to update data. No other user can concurrently access or update the record.
- Use a delete operation when you want to delete a record. No other user can concurrently access or update the record.

Do not use a find operation if any other user or program might corrupt the data. For example, you might retrieve a record with a find operation, make changes, and try to store it only to find another user has deleted the record. To prevent these situations and to avoid processing problems, always lock the record you are updating. (If you do not lock or modify a record before storing it, DMSII returns an exception.)

Preventing Deadlocks

When a record is locked, any other user trying to lock that record must wait until the record is free. However, there are times when a record can get caught in a loop where it is never freed. This situation, shown in the following example, is called a *deadlock*.

User 1

Locks record A

Locks record B (waits for B to be unlocked)

User 2

Locks record B

Locks record A (waits for A to be unlocked)

DMSII prevents deadlocks in one of two ways:

- If one user has a higher priority than the other, DMSII locks the records for the higher priority user and returns an exception to the other user.
- If both users have the same priority, DMSII returns an exception to both users and frees both records. Both users can then redo the task that locked the records.

If a deadlock exception is returned to your application program, relock all records.

Adding New Records to a Data Set

Adding a new record is a three-step process:

1. Use the create operation to move the format and structure of the record into the user work area.

The create operation reserves space in the user work area for the new record. The work area is initialized with the default initial value for each item in the record.

2. Use data transfer operations, such as get, put, and compute, to place values in the items.

The values must conform to the following requirements:

- All key items must have valid values; they cannot be null. Do not use either the COBOL *if null* statement or the COBOL *set to null* extension with key items.

- Unless duplicates are allowed, a key item in a new record cannot have the same value as a key item in an existing record.
 - If a data or key item has specified limits (set in the DASDL by the VERIFY clause), those limits must be met.
3. Use the store operation to place the record into the data set.
The record is not part of the database until it is explicitly stored. The store operation takes the data set record in the user work area, checks that all values conform to requirements, explicitly places the data set record into the database, and makes the appropriate entries in each of the sets and automatic subsets declared in the DASDL for the data set.

If you have an audited database, the create operation can be performed in or out of transaction state. However, for faster processing time, create records and transfer data outside of transaction state. The store operation must always be performed in transaction state.

Changing the Value of Items

The operations used to change the value of an item depend on whether the item was defined as a required item. A *required item* must be assigned a value when it is stored in the data set. For example, a key item is always required. Nonrequired items might or might not have a value when they are stored.

Changing a Nonrequired Item

If the item is not required, you can change the value by using the following program logic:

1. Lock the record.

The lock operation places the record in the user work area and prevents any other program from concurrently updating or deleting the record.

2. Transfer a new value to the item.

The data transfer operations, such as get, put and compute, place the new value in the item. The values must conform to all requirements defined in the DASDL source file.

3. Place the record back in the database using a store operation.

The store operation takes the data set record from the user work area and explicitly stores it in the database. The store operation ensures that the values meet any requirements and makes the appropriate entries in each of the sets and automatic subsets declared in the DASDL for the data set.

Any of the nonrequired items in the record can be changed once the record has been locked. Make all your updates to the record before storing and freeing the record. Any item in the record that is not specifically changed, remains the same.

Note: To prevent a corrupted database and to ensure the correct data is stored, do not reuse the work area until the record is stored.

If your database is not audited, store the record immediately after making all your updates.

If you have an audited database, the lock operation and the transfer of data can be done either in or out of transaction state. When to initiate these operations depends on which DASDL options are set:

- If INDEPENDENTTRANS is not set, process time is faster when you lock records and transfer data outside of transaction state.
- If INDEPENDENTTRANS is set, lock the records and transfer data inside of transaction state.

The store operation must always be performed in transaction state.

Changing a Required Item

When coding program logic to change a required item, ask the following two questions:

- Are duplicate values allowed?
- Is the DASDL option KEYCHANGEOK set for the structure?

If a required item is part of a set in which duplicates are allowed or if the KEYCHANGEOK option is set, the required item can be changed using the same procedure as a nonrequired item. The procedure is described under "Changing a Nonrequired Item" in this section.

However, when duplicates are not allowed and the KEYCHANGEOK option is not set, use the following program logic to change the required item:

1. Delete the record.

The delete operation finds the record, places it in the user work area, and locks the record (to prevent any other program from concurrently updating or deleting the record). The delete operation then deletes the original record from the database.

2. Immediately re-create the record.

The recreate operation preserves the image of the record in the user work area, including the values in the data items.

3. Transfer data into the newly re-created record.

Use the data transfer operations to replace values in any or all items. If a value is not replaced, the value in the image becomes part of the re-created record.

The values must conform to the following requirements:

- All required items must have valid values; they cannot be null. Do not use either the COBOL *if null* statement or the COBOL *set to null* extension with any required item, such as a key item. A null required item causes processing errors when the record is stored.
 - If a data item has specified limits (set in the DASDL by the VERIFY clause), those limits must be met.
4. Store the record in the database.

The record is not part of the database until it is explicitly stored by a store operation. The store operation ensures the values meet the requirements and makes appropriate entries in the data set and in each of the sets and automatic subsets declared in the DASDL source file for the data set.

Caution

After you delete the record, do not perform any operation that affects the user work area until you re-create the record. The delete operation leaves an image of the record format and values in the work area. The recreate operation uses that image to re-create the record. If the work area is changed, you cannot re-create the record.

If you have an audited database, perform all operations—from delete to store—in transaction state.

Deleting Data Records

The delete operation locks and removes the designated record from a data set. All corresponding set and automatic subset entries also are removed. Any current path pointing to the deleted record is invalid because it is pointing to a virtual record.

Notes:

- *There are certain instances when it is not possible to delete a set record, such as when the data set owns an embedded data set. In these cases, you must delete every dependent record before deleting the master record.*
- *If a data set has a manual subset, use the remove operation to delete any corresponding records.*

Adding and Deleting Key Entries in Manual Subsets

With language extensions, you can add and delete key entries in manual subsets.

Use the insert operation to enter key entries in manual subsets. A record can be inserted only once; any attempt to insert a record a second time returns an exception. If DMSII cannot find a corresponding current data set record, the Accessroutines returns an exception to your program.

Use the remove operation to delete key entries in manual subsets. If DMSII cannot find a corresponding current data set record, the Accessroutines returns an exception to your program.

When your program deletes a data set record, use the remove current operation to delete corresponding key entries from a manual subset.

***Note:** The remove current operation does not check to ensure the corresponding data set entry has been deleted.*

Section 8

Recovering an Audited Database

For audited databases, DMSII automatic recovery mechanisms are initiated whenever an abort or a halt/load occurs.

- An *abort* occurs when an update program unexpectedly terminates while in transaction state.
- A *halt/load* occurs when a system-initialization procedure temporarily halts the system and loads the operation system from a disk to main memory.

Note: *If there is a storage media failure, data contained on disks or packs might not be able to be read or written. DMSII data and programs must be recovered using the DMUTILITY program. The DMUTILITY program is described in the DMSII Utilities Operations Guide.*

Each audited database has an associated *audit trail*. The audit trail contains the beforeimages and afterimages of every update operation. DMSII uses information in the audit trail to ensure the database contains only successfully completed transactions.

Each audited database also has a specialized data set called the *restart data set*. The records in the restart data set contain information about the last successfully completed operations that physically changed the database (adding, deleting, or modifying data values).

Every program that updates an audited database must include code that handles recovery after a database abort or halt/load. This section explores the ways your application program works with DMSII to rebuild or recover an audited database after a software or hardware failure. It includes

- A brief explanation of what happens when the ABORT task is initiated
- A brief explanation of what happens when a halt/load occurs
- A description of the DASDL options that affect programming for recovery after an abort or halt/load
- A discussion of how to use the restart data set
- A discussion of how to use data management operations in transaction state to effectively recover an audited database

At the end of the section is an example of a COBOL restartable program with an audited database.

Understanding the ABORT Task

Historically, two conditions cause the Accessroutines to initiate the ABORT task: if a program terminates while it is in transaction state or if a program attempts to close

the database while it is in transaction state. However, with the addition of the DASDL *INDEPENDENTTRANS* option, the ABORT task can be suppressed.

The ABORT task causes DMSII to

- Stop processing any program using the same database as the terminated program.
- Back out all transactions according to the appropriate algorithm. The algorithms depend on what DASDL options are set.
- Discontinue the program that initiated the ABORT task.
- Notify all stopped programs that an abort occurred.

It is the responsibility of each stopped program to resume processing from the correct transaction. Each program uses information in its restart data set to decide where to resume processing.

For details about the ABORT task, read the DMSII Utilities Operations Guide. Read “Setting the DASDL *INDEPENDENTTRANS* Option for Concurrency Control” later in this section for more information about suppressing the ABORT task.

Conventional Aborts

A conventional abort refers to instances where neither the DASDL option *INDEPENDENTTRANS* or *BACKOUTTOSPT* are set. The Accessroutines sequentially scans the audit trail backward. Beforeimages are applied for each transaction – even transactions that were completed successfully – until the Accessroutines detects a quiet point.

Aborts Where *BACKOUTTOSPT* Is Set

When the DASDL option *BACKOUTTOSPT* is set, the Accessroutines sequentially scans the audit trail backward. Beforeimages are applied for each transaction – even transactions that were completed successfully – until the Accessroutines detects a syncpoint.

Read the DMSII DASDL Reference Manual for more details about *BACKOUTTOSPT*.

Understanding a Halt/Load

A halt/load recovery is executed whenever the database stack terminates abnormally. The Accessroutines automatically initiates the DMRECOVERY program to recover the database. The DMRECOVERY program

1. Finds the current end of the audit trail
2. Sequentially scans the audit trail backward until it finds a quiet point
3. Moves back two controlpoints from the quiet point
4. Goes forward, applying afterimages until the end of the audit trail
5. Goes backward, applying beforeimages until the quiet point

6. If REAPPLYCOMPLETED is set, resubmits completed transactions

All data management jobs wait until recovery is complete. It is the responsibility of each stopped program to resume processing from the correct transaction. Each program uses information in its restart data set to decide where to resume processing.

If the DASDL options INDEPENDENTTRANS and REAPPLYCOMPLETED are set, all completed transactions are reapplied during a halt/load recovery. Read "Setting the DASDL INDEPENDENTTRANS Option for Concurrency Control" and "Setting the DASDL REAPPLYCOMPLETED Option for Recovery" later in this section.

If the DMRECOVERY program encounters a fatal error, it terminates. If the fatal error indicates that the audit file is corrupted, the contents of the bad block of that audit file are printed out.

For details about the DMRECOVERY program, read the DMSII Utilities Operations Guide.

Defining Recovery Options through DASDL

Before you begin coding your update program, examine the DASDL definition of the database. At a minimum, an audited database must have a restart data set and the AUDIT option must be set. Recovery techniques can be specified using the DASDL options BACKOUTTOSPT, INDEPENDENTTRANS, RDSSTORE, REAPPLYCOMPLETED, and RESTART TRANSACTION ONLY. In addition, the physical attributes of the audit trail can be tailored using the AUDIT TRAIL declaration.

Table 8-1 briefly describes the DASDL recovery options. It is followed by a discussion of the AUDIT, INDEPENDENTTRANS, and REAPPLYCOMPLETED options.

The effect of the BACKOUTTOSPT option is briefly explained under "Aborts Where BACKOUTTOSPT Is Set." The effect of the RDSSTORE option is briefly explained under "Using the AUDIT Option to Write Records to the Restart Data Set." The effect of the RESTART TRANSACTION ONLY option is briefly explained under "Adding and Deleting Records in the Restart Data Set."

For a complete explanation of all the DASDL options and declarations, read the DMSII DASDL Reference Manual.

Table 8-1. DASDL Options for Audited Databases

Option	Purpose
AUDIT	Causes the database to be audited. Images are written to the audit trail. This option can also include syncpoint and controlpoint specifications.
BACKOUTTOSPT	Causes all transactions since the latest syncpoint to be unconditionally backed out.

continued

Table 8-1. DASDL Options for Audited Databases (cont.)

Option	Purpose
INDEPENDENTTRANS	Ensures each logical transaction is independent of every other logical transaction. The application program can perform single transaction aborts and can use savepoints.
RDSSTORE	Tells the Accessroutines to store information in the restart data set at both begin and end transaction operations.
REAPPLYCOMPLETED	Causes all completed transactions in the audit trail to be reapplied during halt/load recoveries. Each restart area reflects the last completed transaction for each program. This option is valid only if INDEPENDENTTRANS is set also.
RESTART TRANSACTION ONLY	Ensures that only tasks which are abnormally terminated within transaction state have restart records. If a program terminates abnormally outside transaction state, there is no restart record.

Setting the DASDL AUDIT Option for Audited Databases

By definition, the DASDL *AUDIT* option must be set for a database to be audited. The *AUDIT* option causes the Accessroutines to automatically write information to the audit trail each time DMSII executes a data management operation in transaction state that changes the database.

The audit trail contains various control records, a sequence of beforeimages and afterimages, and index tables. The audit trail can be written to tape or disk and can be used as input data during database recovery.

The audit trail is used by the DMSII software to

- Recover the database after a halt/load. The DMRECOVERY program uses afterimages to reapply transactions, and uses beforeimages to back out transactions.
- Provide restart information to user programs.
- Reconstruct portions of the database that have been lost because of hardware errors.
- Back out aborted transactions. The ABORT task uses beforeimages to back out transactions.
- Roll back the entire database to a user-specified point.
- Rebuild the entire database to a user-specified point using DMUTILITY.

Note: Store the audit trail either on a different pack from the database or on tape. Otherwise, if the database pack is damaged, the audit trail cannot be used.

Setting the DASDL INDEPENDENTTRANS Option for Concurrency Control

The DASDL *INDEPENDENTTRANS* option forces DMSII to use a two-phase locking concurrency control. Concurrency control prevents two or more simultaneous updates to the same record by serializing the updates.

Two-phase locking means a record must be locked before a user can modify the record. Records can be locked within transaction state. Until the transaction for that record is complete, the record cannot be freed. All other users or programs must wait until the record is freed before attempting to modify the record. If the record is locked within transaction state, only an end transaction operation can free the record.

If a program aborts in transaction state, all the records the program is currently updating are locked – ensuring that no other program could be updating or accessing the records. Therefore, no other program can be affected. There is no need for an ABORT task recovery. DMSII simply backs out the transactions for the aborted program while all other programs continue with their normal processing.

Setting the DASDL REAPPLYCOMPLETED Option for Recovery

The DASDL *REAPPLYCOMPLETED* option can only be set if the DASDL *INDEPENDENTTRANS* option is set. When the *REAPPLYCOMPLETED* option is set, a transaction cannot be backed out once an end transaction operation is successfully completed.

Recovery occurs in the following phases:

- Halt/load recovery recovers to the syncpoint.
- All completed transactions are reapplied.

Once a transaction is audited, the *REAPPLYCOMPLETED* option guarantees that even in the event of a halt/load, the changed records are stored in the database.

Understanding the Purpose of a Restart Data Set

A restart data set identifies the point from which a program can restart the processing of the database after a processing failure.

Each audited database must have one, and only one, restart data set. The restart data set must be disjoint. No links, control items, or embedded sets can be declared in the restart data set, and no links can point to it. Related subsets must be automatic.

The DASDL reserved word *RESTART* identifies the data set as a restart data set. DASDL automatically assigns two data items to every restart data set: an item that counts the number of times a program is in transaction state and an item that identifies the source of the restart record. In addition to these automatic items, most restart data sets also include an item that identifies the last successful transaction and a program identifier.

Identifying the Last Successful Transaction

At a minimum, the restart data set must contain information that uniquely identifies where the program was when the program terminated abnormally or a halt/load occurred.

One item in the restart data set record should identify the last successful transaction or the last unsuccessful transaction executed by your program. Either transaction can be used to mark where the program should restart. For example, if your program is the only one accessing the database and its task is to update last names, your restart data set record might contain the last successfully updated last name, and no other data.

However, there are instances when the restart data set record might need to have more than one item of information. For example, if your program is updating salaries, the restart data set record might need to contain both the employee number and salary to uniquely identify the last successful update.

Identifying Your Program

If more than one program can concurrently access the database, the restart data set record should have one item that uniquely identifies the program.

If you do have program identifiers, the database definition should include an index set where duplicates are allowed for the program identifiers. This arrangement ensures that the ABORT task and the DMRECOVERY program will not fail if the values in key items are the same.

Example of the DASDL Source Code for a Restart Data Set

The following restart data set has two data items: one for the program identifier, and one for the last successful transaction:

```
STARTAGAIN RESTART DATA SET
  (STARTAGAIN-PRGM-ID      ALPHA(25);
   STARTAGAIN-SUCTRAN-INFO ALPHA(250);
  );
```

Your program can store the name of the program that was working on the database during recovery in the data item STARTAGAIN-PRGM-ID. The program identifier cannot be more than 25 alphabetic characters.

Your program can also store information about where to restart processing in the data item STARTAGAIN-SUCTRAN-INFO. This data item can be a maximum of 250 alphabetic characters.

Adding and Deleting Records in the Restart Data Set

Because the restart data set is part of the database, application programs can easily access and use information in the restart data set to reinitialize the database.

DMSII treats the restart data set the same as any other data set. Each time your program invokes the restart data set, DMSII reserves a user work area for the data set. Your application program is responsible for creating restart data set records, placing data into the records, and storing the records. You can use the same operations that update any data set.

To guarantee the best recovery results, make sure you store one restart record for each transaction. You can do this by explicitly storing the record or by using the AUDIT option of the begin transaction operation or the end transaction operation. Read "Using the AUDIT Option to Write Records to the Restart Data Set" for more information about this option.

If your program is discontinued, DMSII automatically writes a restart record to your restart data set. If your program is aborted, DMSII automatically writes a restart record to your restart data set and writes the appropriate restart data set record to each concurrently running update program for the database.

Note: *If the DASDL RESTART TRANSACTION ONLY option is set, DMSII writes a record to the restart data set only if your program terminates abnormally within transaction state. It does not write a record if the program is discontinued, or if the program is aborted outside transaction state.*

However, DMSII does not automatically delete existing records in the restart data set. If your program performs a large number of transactions, the restart data set can contain a large number of records. To preserve storage space and to maintain the most accurate records, each user program should periodically delete records from its restart data set.

For example, if all of your end transaction operations use the AUDIT option, each operation causes DMSII to add a new record to the restart data set. After each end transaction operation you should find and delete the prior restart record.

Knowing When to Read the Restart Data Set Records

There are two processing times when your program should examine the restart data set to locate restart information for your program:

- Immediately after the database is opened for update
- When the program is restarted by a recovery process

In both cases, use a find operation to search the restart data set for a restart record for your program.

If there is no record in the restart data set for your program, all previous database activities completed successfully. Your program code should continue the program.

If there is a record in the restart data set with your program identifier, your program should initiate restart code.

Identifying the Restart Procedure Tasks

The restart code needs to do the following three tasks in order to resume processing:

1. Extract the data in the restart data set record.
2. Reposition all non-DMSII files.
3. Delete the restart record.

If, after an abort or a halt/load, your program cannot find a restart record, you need to restart the program from the beginning. Therefore, also include restart code that resumes processing from the beginning rather than the restart point noted in the restart data set record.

You can also include code that forces your job summary to note whether a restart occurred.

For an example of restart code, read "Example of a Restartable Program with an Audited Database" later in this section.

Using Transaction State

Your application program should enter transaction state whenever the program code updates the database. The begin transaction operation causes the program to enter transaction state. When the program code has completed the logical transaction, use an end transaction operation to leave transaction state.

The following data management operations, noted in alphabetical order, change the values stored in the database and should be performed within transaction state, as part of a logical transaction:

Operation	Explanation
Assign	Establishes a link from one data set record to another
Assign No	Deletes a link from one data set record to another
Delete	Removes a record from a data set and removes references to the record in all associated sets and automatic subsets
Generate	Creates a subset from one or more manual subsets
Insert	Places a record into a manual subset
Remove	Deletes a record from a manual subset
Store	Places a new or modified record into a data set or the global record

Initiate these operations only while your program is in transaction state so that

- Audit records for the modification are between a begin and an end transaction operation.
- Recovery routines can find a point in the audit trail where no partially completed transactions are stored in the database.

You can initiate some data management operations, such as find, lock, or create both in and out of transaction state. Check to see what DASDL options are set to decide when

to initiate the operations. For example, if `INDEPENDENTTRANS` is set, lock records within transaction state. Stay in transaction state to transfer data and store the record. If `INDEPENDENTTRANS` is not set, you might decide to have your program lock a record and transfer data outside transaction state and then enter transaction state to store the record.

Always initiate the open and close database operations out of transaction state.

Beginning and Ending Transaction State

When a begin transaction operation is executed, the stack associated with the program and the physical database enter transaction state. A beforeimage of the record is written to the audit trail.

The restart data set must be invoked by each begin transaction operation. The begin transaction operation must note the name of the restart data set as part of its syntax.

While in transaction state, the program can perform the desired operations. If an operation returns a record to the database, changed or new, an afterimage of the record is written to the audit trail when an end transaction is successfully executed.

***Note:** If a record is deleted, only its beforeimage exists in the audit trail. If a record is added, only its afterimage exists in the audit trail.*

The database remains in transaction state until the same program issues and successfully executes an end transaction operation.

If the DASDL `INDEPENDENTTRANS` option is set, all database records that were locked by the program (either in or out of transaction state), are unlocked when an end transaction operation is completed. Any attempt to unlock a record within transaction state is ignored.

If `INDEPENDENTTRANS` is not set, you can free records within transaction state using data management operations that implicitly or explicitly free records (create, delete, find, free, lock, modify, recreate, secure, and store). For example, if you lock and then store a record, the store operation implicitly frees the record.

If an error occurs when either a begin or an end transaction operation is executing, DMSII forces the program to leave transaction state.

Using the AUDIT Option to Write Records to the Restart Data Set

To write a restart record to the restart data set, your program must

- Execute a create operation to reserve the user work area for the restart data set record.
- Enter transaction state and identify the restart data set.
- Process records.
- Exit transaction state.

Both the begin and end transaction operations have an AUDIT option. The AUDIT option instructs DMSII to capture information for the restart data set in the user work area reserved by the create operation. Each time DMSII executes a begin or an end transaction operation, it checks to see if the AUDIT option is set. Then, depending on how you code your begin and end transaction operations, you can instruct DMSII to write the captured information as a record to the restart data set

- Only when a begin transaction operation is executed.
- Only when an end transaction operation is executed.
- At both a begin transaction and an end transaction operation.
- At neither point. If the AUDIT option is set at begin transaction and reset at end transaction, the captured restart information is not written to the restart record.

By default, a begin transaction operation always sets the AUDIT option. In contrast, you must explicitly set the AUDIT option for an end transaction operation.

If you audit only the begin point, you have no guarantee that the previous end transaction operation completed successfully. For more accurate record keeping, use the AUDIT option with all end transaction operations.

Be consistent in how you audit the database. For example, do not audit both the begin and end points for one transaction, only the beginning point for another, and only the ending point for still another.

Note: If the DASDL RDSSTORE option is set, restart records are not created at recovery time. A restart record is written to the restart data set at both begin and end transaction points.

Forcing a Syncpoint

A quiet point is a time when no program or user is updating a database. It is a moment when the database is stable. Each quiet point is marked by the Accessroutines as a recovery checkpoint. If a transaction fails, the DMSII DMRECOVERY program backs out all transactions until the last quiet point.

A syncpoint is a forced quiet point. The syncpoint option in DASDL specifies the number of transactions that should be successfully executed before the Accessroutines forces a syncpoint. By using the SYNC option in an end transaction operation, your program can override the DASDL specification.

The SYNC option of the end transaction operation creates a syncpoint after the transaction is successfully completed. When the SYNC option is specified, the end transaction operation should be immediately followed by recovery code. For example,

```
END-TRANSACTION <audit option> <restart data set name> SYNC
ON EXCEPTION <statement>
```

However, since syncpoints decrease system throughput, use the SYNC option sparingly.

Note: *The DASDL REAPPLYCOMPLETED option forces an audit after each end transaction operation. This means that each end transaction operation is a syncpoint. However, unlike the SYNC option, the REAPPLYCOMPLETED option does not cause the system to wait for other users to leave transaction state.*

Multitasking in Transaction State

Once a process places an internal database in transaction state, any dependent processes that can address the database globally can perform operations against the database. This arrangement of a parent process and one or more dependent tasks is known as *multitasking*.

A dependent process can participate in the transactions of the parent process or execute its own independent transactions with one exception—the process that executes a begin transaction operation must be the process that executes the end transaction operation.

Any multitasking process that locks records or that is in transaction state should not initiate any nondata management operation that causes the process to wait. DMSII deadlock analysis cannot resolve deadlocks involving operations that are not data management operations.

Example

In the following example, two programs access the same database. Program 1 is in transaction state, making it possible for Program 2 to perform an update operation by calling the STOREEMP procedure. Program 1 then ends transaction state. Neither program is in conflict at any time.

Program 1

BTR;

ETR;

Program 2

STOREEMP; % LEGAL

Understanding Language Extension Recovery Commands

The data management operations noted in Table 8-2 are supported only through language extensions. These operations are valid if the DASDL *INDEPENDENTTRANS* option is set for the database. If the *INDEPENDENTTRANS* option is not set, any attempt to abort or cancel a transaction can corrupt the entire database.

Table 8-2. Language Extensions Auditing Operations

Operation	Brief Explanation
Abort transaction	Discards all updates made in a transaction and brings the database back to the beginning of the transaction. The program and all concurrent programs continue running. All other transactions are processed.
Cancel transaction point	Discards all updates made in a transaction back to a specified savepoint and brings the database back to the savepoint. The program remains within the same transaction state. The program and all concurrent programs continue running.
Savepoint	Writes a savepoint record to the audit trail, which serves as an option marker for partially aborting a transaction. The savepoint applies only to the current transaction, and does not affect halt/load recovery.
Terminate	Initiates DMSII fault termination of the program for an exception.

The cancel transaction point operation partially backs out a transaction to a user-specified savepoint. Your program can have one or more savepoints. For example, if one transaction within your program writes data to a general ledger and to separate journals, you might identify a savepoint at the end of the general ledger process and a savepoint at the end of each journal process. Then, depending on the error condition which initiates the cancel transaction point operation, the transaction is backed out to a specified savepoint.

Consult the appropriate programming language reference manual for syntax of the language extensions, and for their availability.

Example of a Restartable Program with an Audited Database

The following DASDL source file defines the restart data set used in the COBOL example of a restartable program.

The COBOL program is not presented as an example of the best way to code a program. It is designed to highlight the restartable aspects of programming an application. The intent of restart code is to skip the blocks that have already been processed.

The program reads an input transaction file. Each record contains enough information to update one member of the PERSONNEL data set in the COMPANYX database. All the operations involved in actually updating one member are grouped into a logical transaction.

DASDL Source for a Restartable Program

```

OPTIONS (AUDIT);

RESTARTAREAS RESTART DATA SET (
    RESTARTALPHAID    ALPHA (18);
                    % UNIQUE ID FOR EACH PROGRAM
    RESTARTNUMID      NUMBER (12);
                    % TO DISTINGUISH MULTIPLE COPIES OF PROGRAMS
    USERINFO          ALPHA (180) INITIALVALUE LOW-VALUE;
                    % INFORMATION PROGRAMS NEED TO RESTART
);

RESTARTSET SET OF RESTARTAREAS KEY IS (RESTARTALPHAID, RESTARTNUMID);

PERSONNEL DATA SET (
    EMP-NO            "EMPLOYEE NUMBER" NUMBER (4);
    EMP-DATA          "EMPLOYEE DATA" NUMBER (4);
);

EMPLOYEE SET OF PERSONNEL KEY EMP-NO;

AUDIT TRAIL ATTRIBUTES
    (BLOCKSIZE = 150,
    KIND = TAPE VERIFY);
RESTARTAREAS (POPULATION = 100);
RESTARTSET (POPULATION = 100);
PERSONNEL (POPULATION = 150);
EMPLOYEE (POPULATION = 150);

```

Example of a COBOL Restartable Program

```

000100 $ SET BDMS LIST FREE
000200 IDENTIFICATION DIVISION.
000300 ENVIRONMENT DIVISION.
000400 INPUT-OUTPUT SECTION.
000500 FILE-CONTROL.
000600* TF IS THE INPUT TRANSACTION FILE.
000700* TF MAY BE LABEL EQUATED TO TAPE, IF DESIRED.
000800     SELECT TF ASSIGN TO DISK,
000900     ACCESS MODE IS SEQUENTIAL,
001000     ACTUAL KEY IS TFRECNUM.
001100     SELECT PRINT ASSIGN TO PRINTER.
001200 DATA DIVISION.
001300 FILE SECTION.
001400 FD TF.
001500 01 BLOCKBUF.
001600     02 B-NUM      PIC 9(6)    COMP.
001700*     TAPE BLOCK NUMBER - - STARTS AT 1
001800     02 E-NUM      PIC 9(4)    COMP-2.
001900*     EMPLOYEE NUMBER
002000     02 E-DATA     PIC 9(4)    COMP-2.

```

Recovering an Audited Database

```

002100*      EMPLOYEE DATA
002200      02 FILLER    PIC X(162).
002300 FD PRINT.
002400 01 PRINTBUF.
002500      02 PB1      PIC X(19).
002600      02 PB2      PIC 9(4).
002700      02 PB3      PIC X(11).
002800      02 PB4      PIC 9(4).
002900      02 PB5      PIC X(4).
003000      02 PB6      PIC 9(4).
003100      02 FILLER    PIC X(86).
003200 DATA-BASE SECTION.
003300 DB COMPANYX.
003400 01 PERSONNEL.
003410 * 01 PERSONNEL STANDARD DATA SET(#4).
003420 *      EMPLOYEE SET(#5,AUTO) OF PERSONNEL KEY IS EMP-NO.
003430 *      02 EMP-NO PIC 9(4) COMP-2.  "EMPLOYEE NUMBER"
003440 *      02 EMP-DATA PIC 9(4) COMP-2.  "EMPLOYEE DATA"
003500 01 RESTARTAREAS.
003510 * 01 RESTARTAREAS RESTART DATA SET(#2).
003520 *      RESTARTSET SET(#3,AUTO) OF RESTARTAREAS
003530 *      KEYS ARE RESTARTALPHAID,RESTARTNUMID.
003540 *      02 RESTARTALPHAID PIC X(18) DISPLAY.
003560 *      02 RESTARTNUMID PIC 9(12) COMP-2.
003570 *      02 USERINFO PIC X(180) DISPLAY.
003600 WORKING-STORAGE SECTION.
003700 77 I          PIC 9(5)    COMP-1    VALUE 0.
003800 77 TFRECNUM    PIC 9(11)   COMP-1    VALUE 0.
003900 01 WORK-USERINFO.
004000      02 BLOCKSIN          PIC 9(6)    COMP    VALUE 0.
004100      02 GOOD-TRANSACTIONS PIC 9(6)    COMP    VALUE 0.
004200      02 BAD-TRANSACTIONS  PIC 9(6)    COMP    VALUE 0.
004300      02 FILLER          PIC X(162).
004400 01 PRINT-LINE.
004500      02 FILLER          PIC X(20)    VALUE "GOOD TRANSACTIONS = ".
004600      02 GOOD-TRANS      PIC 9(6).
004700      02 FILLER          PIC X(5)    VALUE SPACES.
004800      02 FILLER          PIC X(19)    VALUE "BAD TRANSACTIONS = ".
004900      02 BAD-TRANS      PIC 9(6).
005000      02 FILLER          PIC X(76)    VALUE SPACES.
005050 01 TRANS-STATUS  PIC X          VALUE "F".
005070      88 WE-ABORTED                                VALUE "T".
005100 PROCEDURE DIVISION.
005200 THE-ONLY SECTION.
005300 INITIALIZE.
005400      OPEN INPUT TF.
005500      OPEN OUTPUT PRINT.
005600      OPEN UPDATE COMPANYX.
005700      CREATE RESTARTAREAS.
005800      MOVE "UPDATE/COMPANYX" TO RESTARTALPHAID.
005900      MOVE MYSELF(JOBNUMBER) TO RESTARTNUMID.
006000      IF MYSELF(TASKVALUE) > 0 GO TO RESTARTED.

```

```

006100      IF MYJOB(RESTARTED) = VALUE TRUE GO TO HALTLOADED.
006200
006300 MAINLOOP.
006400      READ TF AT END
006500          PERFORM WRAPUP
006600          STOP RUN.
006700      PERFORM A-TRANSACTION THRU A-TRANSACTION-EXIT.
006750      If WE-ABORTED GO TO RE-INITIATE
006800      GO TO MAINLOOP.
006900
007000 A-TRANSACTION.
007100*      ENSURE ALL BLOCKS ARE PRESENT ON TF.
007200      IF B-NUM NOT = TFRECNUM PERFORM DIE.
007300      MOVE ALL SPACES TO PRINTBUF.
007400      MOVE "UPDATING EMPLOYEE" TO PB1.
007500      MOVE E-NUM TO PB2.
007600      MOVE "DATA FROM" TO PB3.
007700      MOVE "TO" TO PB5.
007800      MOVE E-DATA TO PB6.
007900      MODIFY EMPLOYEE AT EMP-NO = E-NUM ON EXCEPTION
008000          IF DMSTATUS(DEADLOCK)
008100              GO TO A-TRANSACTION
008200          ELSE
008300              IF DMSTATUS(NOTFOUND)
008400                  ADD 1 TO BAD-TRANSACTIONS
008500                  GO TO A-TRANSACTION-EXIT
008600              ELSE
008700                  CALL DMTERMINATE.
008800      MOVE EMP-DATA TO PB4.
008900      MOVE E-DATA TO EMP-DATA.
009000*      AT THIS POINT, ALL INPUT I/O HAS BE DONE.
009100*      ALL RECORDS HAVE BEEN FOUND AND/OR LOCKED,
009200*      ALL DECISIONS HAVE BEEN MADE,
009300*      AND ALL DATA HAS BEEN MOVED.
009400*      ONLY CHANGES TO THE DATABASE AND OUTPUT I/O REMAIN
009500*      TO BE DONE IN TRANSACTION STATE.
009600*      A LOOP BACK TO A-TRANSACTION IS MADE IN CASE OF
009700*      DEADLOCK -- THIS IS POSSIBLE BECAUSE NO CHANGES
009800*      TO THE DATABASE HAVE YET BEEN MADE.
009900      BEGIN-TRANSACTION NO-AUDIT RESTARTAREAS ON EXCEPTION
010000          IF DMSTATUS(ABORT)
010100              GO TO ABORTED
010200          ELSE
010300              IF DMSTATUS(DEADLOCK)
010400                  GO TO A-TRANSACTION
010500              ELSE
010600                  CALL DMTERMINATE.
010700      STORE PERSONNEL.
010800*      OUTPUT FILES OTHER THAN DATABASE FILES AND PRINTER
010900*      FILES WOULD BE WRITTEN WITH DIRECT I/O HERE.
011000      WRITE PRINTBUF.
011100      ADD 1 TO GOOD-TRANSACTIONS.

```

Recovering an Audited Database

```
011200      MOVE TFRECNUM TO BLOCKSIN.
011300      MOVE WORK-USERINFO TO USERINFO.
011400      END-TRANSACTION AUDIT RESTARTAREAS ON EXCEPTION
011500          IF DMSTATUS(ABORT)
011600              GO TO ABORTED
011700          ELSE
011800              CALL DMTERMINATE.
011900 A-TRANSACTION-EXIT.
012000      EXIT.
012100
012200 WRAPUP.
012300      MOVE ALL SPACES TO PRINTBUF.
012400      MOVE "WRAPUP" TO PB1.
012500      WRITE PRINTBUF.
012600      CLOSE COMPANYX ON EXCEPTION
012700      IF DMSTATUS(ABORT)
012900          GO TO ABORTED
012950      ELSE
012960          CALL SYSTEM DMTERMINATE.
013000      MOVE GOOD-TRANSACTIONS TO GOOD-TRANS.
013100      MOVE BAD-TRANSACTIONS TO BAD-TRANS.
013200      WRITE PRINTBUF FROM PRINT-LINE.
013300
013400 ABORTED.
013500      MOVE ALL SPACES TO PRINTBUF.
013600      MOVE "ABORTED" TO PB1.
013700      WRITE PRINTBUF.
013800      CLOSE TF.
013900      OPEN INPUT TF.
013950      MOVE "T" TO TRANS-STATUS.
014000      GO TO A-TRANSACTION-EXIT.
014100
014200 HALTLOADED.
014300      SET MYJOB(RESTARTED) TO VALUE FALSE.
014400      MOVE ALL SPACES TO PRINTBUF.
014500      MOVE "HALT/LOADED" TO PB1.
014600      WRITE PRINTBUF.
014700      GO TO RE-INITIATE.
014800
014900 RESTARTED.
015000      MOVE MYSELF(TASKVALUE) TO RESTARTNUMID.
015100      MOVE ALL SPACES TO PRINTBUF.
015200      MOVE "RESTART RUN" TO PB1.
015300      WRITE PRINTBUF.
015400      GO TO RE-INITIATE.
015500
015600 RE-INITIATE.
015700      LOCK RESTARTSET AT
015800          RESTARTALPHAID = RESTARTALPHAID AND
015900          RESTARTNUMID = RESTARTNUMID
016000      ON EXCEPTION
016100          IF DMSTATUS(NOTFOUND)
```

```

016200* NO TRANSACTIONS COMPLETED AGAINST THE DATABASE
016300* PRIOR TO THE RESTART POINT.
016400      CLOSE TF
016500      OPEN INPUT TF
016600      GO TO MAINLOOP
016700      ELSE
016800      CALL DMTERMINATE.
016900* A RESTART RECORD IS PRESENT. RETRIEVE THE RESTART
017000* INFORMATION, DELETE THE RESTART RECORD, AND
017100* POSITION THE INPUT FILE USING THE VALUE IN
017200* BLOCKSIN.
017300      MOVE USERINFO TO WORK-USERINFO.
017400      BEGIN-TRANSACTION RESTARTAREAS.
017500      DELETE RESTARTAREAS.
017600      END-TRANSACTION RESTARTAREAS.
017700
017800 POSITION-TF.
017900      COMPUTE TFRECNUM = BLOCKSIN + 1.
018000      SEEK TF.
018100* OUTPUT FILES OTHER THEN DATABASE FILES AND PRINTER
018200* FILES WOULD BE POSITIONED HERE (AND IF DISK OR PACK,
018300* WOULD HAVE THEIR END-OF-FILE ESTABLISHED).
018400      GO TO MAINLOOP.
018500
018600 DIE.
018700      COMPUTE I = I / 0.

```


Section 9

Handling Exception and Error Conditions

As your application program runs, DMSII monitors all database processing. The Access routines return any database processing exceptions and errors to your application program.

DMSII has divided exceptions and errors into 21 major categories. Each category deals with a general and a specific cause for the condition. Appendix A, "DMSII Exceptions and Errors," lists each condition by category.

For each data management operation, you can choose to have your program either

- Check the processing messages for exceptions or errors, and decide on an appropriate course of action.
- Ignore processing messages and have DMSII handle the condition by discontinuing the program. This process is referred to as *masking* the message.

Distinguishing between Exceptions and Errors

An *exception* is internal to the program. It is caused by a problem within the application program, such as an improper selection expression. Other programs, unless they use the same logic that caused the problem, are not affected.

An *error* is external to the program. It is caused by a problem within the database, such as a corrupted data structure, or within the DMSII software, such as corrupted code. All programs accessing the same data structure or the same code receive the same message.

An exception or error can be either fatal or nonfatal.

Fatal Conditions

For fatal conditions, you can choose to document the condition and then terminate, or you can have DMSII terminate the stack.

A fatal exception forces you to exit the program to correct the condition – for example, if the program cannot open the database.

A fatal error causes the database stack to terminate. All fatal errors are reported under the category SYSTEMERROR. For more details read "Understanding SYSTEMERROR Causes and Responses" in this section.

Nonfatal Conditions

For nonfatal conditions, you can include code within your application program to handle the condition.

A nonfatal exception can be corrected without exiting the program. For example, if the current path points to a virtual record, your program can reset the current path.

A nonfatal error allows the program to continue as long as the program does not access corrupted control information. All nonfatal errors are reported under the category `INTEGRITYERROR`. For more details read "Understanding `INTEGRITYERROR` Causes and Responses" in this section.

Understanding `SYSTEMERROR` Causes and Responses

In general, the `Accessroutines` reports a fatal error for one of the following reasons:

- An irrecoverable I/O error exists in the control information used to describe or access user data.
- The `Accessroutines` detects inconsistent control information.
- The `Accessroutines` code contains faults.
- The operation exceeds a system resource limitation.
- An operator discontinued the database.

For fatal errors, the `Accessroutines` returns a `SYSTEMERROR` condition to all application programs that currently have the database open. After all programs have exited the database, `DMSII` terminates the database stack.

The `Accessroutines` then displays a message describing the type of error. This message contains the sequence number of the line in `SYMBOL/ACCESSROUTINES` in which the error was detected. If the error is related to a particular structure, the structure number also is displayed.

The message appears under the mix number of the application program in whose stack the `Accessroutines` was running when the error was detected. If appropriate, `DMSII` initiates a program dump under the `TASKFILE` attribute of the application program.

Understanding `INTEGRITYERROR` Causes and Responses

When a nonfatal error is detected, the `Accessroutines` initiates a program dump and displays a message showing the job and task mix numbers of the application program that detected the error. An `INTEGRITYERROR` condition is returned to programs that detect nonfatal errors.

Application programs can continue to access database structures in which errors have been found as long as they do not access the corrupted control information. Every program that tries to use corrupted information causes a program dump and receives an `INTEGRITYERROR` exception.

Deciding Whether to Mask Messages

The Accessroutines can return an exception or error message for each data management task. As you design your application, look at each of your data management operations to determine if you want to

- Know whether it executed successfully.
- Pinpoint the problem.
- Develop application program code to resolve the problem.
- Know whether the returned condition is fatal.

Based on these considerations, you can decide if you want to **examine** the message or to **ignore** the message.

If you choose to examine the message, use either the DMSTATUS register (for language extensions) or the DBSTATUS exception word (for entry points) to return the message to your application program. Both DMSTATUS and DBSTATUS document the success of the operation, the category and subcategory for the condition, and any corrupted structure.

If you decide to ignore the condition, DMSII automatically handles the condition.

Notes:

- *If the error or exception is fatal, the database and program must be discontinued. The Accessroutines overrides any application program logic and terminates the database stack.*
- *If your application program updates a database for which the DASDL INDEPENDENTTRANS option is not set, you must handle any deadlock condition within your program.*

Handling Exception or Error Conditions through DMSII

If you choose to have DMSII handle the exception or error, the Accessroutines will

- Issue a message. The message appears on the terminal where the program was initiated and on the job printout. The message identifies the cause of the problem by category, subcategory, and when needed, structure number.
- Discontinue the program.
- For fatal conditions, terminate the database stack.

Handling Exception or Error Conditions through Your Program

If you choose to handle the exception or error in your program, include programming logic that

- Examines DMSTATUS or DBSTATUS to determine the category, error type, and structure.
- Decides and initiates one of the courses of action listed below:
 - Execute problem-solving code. Only continue your program if circumventing the error condition will not cause future processing problems or database corruption problems.
 - Return control to DMSII in order to discontinue the program and to terminate the database stack.

Note: There are certain fatal errors and exceptions that cannot be circumvented, such as a corrupted database. DMSII software does not allow the program to continue.

Read Appendix A, "DMSII Exceptions and Errors," for more information on errors, exceptions, and recovery.

Examining the DMSTATUS Register

The DMSTATUS register has four fields:

- DMERROR is a Boolean value that shows an error has happened.
- DMCATEGORY is a numeric value that describes the kind of error.
- DMERRORTYPE is a numeric value that describes the subcategory of the error.
- DMSTRUCTURE is the number of the structure that caused the error.

You can access all fields at once with the DMRESULT operation.

Programming examples can be found in Appendix C, "Sample Programs Using Language Extensions." Consult the language reference manuals for details about how to use the DMSTATUS register. The DMSII DASDL Reference Manual explains the register and each of the fields.

Examining the DBSTATUS Exception Word

Three entry points can be used to determine the content of an exception word. DBSTATUS is examined to determine whether an exception or error has occurred. If an exception or error has occurred, other entry points are used to describe the condition.

- DBSTATUS returns the exception word to the program.
- DBEXCEPTIONNAME returns the name of the exception.
- DBEXCEPTIONTEXT returns the category name, the structure name and number, and the text of the message.

Programming examples can be found in Appendix B, "Sample Programs Using the Interpretive Interface." Consult the DMSII Interpretive Interface Programming Manual for details about how to use DBSTATUS.

Appendix A

DMSII Exceptions and Errors

The Accessroutines returns exception and error messages to an application program when data management operations cannot be successfully or correctly completed.

Exceptions are caused by problems within the application program. For example, an exception is returned if your application program tries to nest logical transactions. Errors are caused by problems within the database or DMSII software. For instance, an error is returned if there is an invalid index within the DMSII code.

Many responses to error messages require running DMSII utilities such as REORGANIZATION, DBCERTIFICATION, DMUTILITY, and REBUILD. For details about the DMSII utilities, consult the DMSII Utilities Operations Guide. Some responses require changes to the DASDL source file for your database. For in-depth information on DASDL, read the DMSII DASDL Reference Manual.

Categorizing Exceptions and Errors

The exceptions and errors are divided into 21 major categories. Each major category has an associated category number and mnemonic. For example, the major category known by the mnemonic SYSTEMERROR is also known as major category 7. Table A-1 lists the numbers and mnemonics for the major categories by category number, in ascending order.

Table A-1. Major Categories for Exceptions and Errors by Category Number

Number	Mnemonic	Number	Mnemonic
1	NOTFOUND	11	OPENERERROR
2	DUPLICATES	12	CLOSEERROR
3	DEADLOCK	13	NORECORD
4	DATAERROR	14	INUSE
5	NOTLOCKED	15	AUDITERERROR
6	KEYCHANGED	16	ABORT
7	SYSTEMERROR	17	SECURITYERROR
8	READONLY	18	VERSIONERROR
9	IOERROR	19	FATALERROR
10	LIMITERROR	20	INTEGRITYERROR
		21	INTLIBERROR

Note: Any message that can be returned through the language extensions can be returned through the interpretive interface. However, INTLIBERROR (category 21) messages can be returned only to application programs using the interpretive interface.

Within each major category, each cause for the exception or error is a subcategory. Every subcategory has an associated subcategory number. Tables A-2 through A-21 list the major categories alphabetically by mnemonic. Under each category, the table lists the valid subcategories and their meaning.

ABORT Subcategories

ABORT is major category 16. The most probable cause for an ABORT message is a database abort or a rollback.

Table A-2. ABORT Subcategories

Subcategory Number	Meaning
1	The database was aborted and rolled back.
2	An abort occurred, and the Transaction Processing System (TPS) rerun was not finished.

AUDITERROR Subcategories

AUDITERROR is major category 15. The most probable cause for an AUDITERROR message is a conflict in transaction state.

Table A-3. AUDITERROR Subcategories

Subcategory Number	Meaning
1	The program attempted to execute a begin transaction operation on a database that is already in transaction state.
2	The program attempted to execute an end transaction operation on a database that is not in transaction state.
3	The program attempted to execute a MIDTRANSACTION statement on a database that is not in transaction state.
4	The program attempted to execute a normal begin or end transaction operation on a database that was opened by TPS.
5	The program attempted to execute a TPS begin transaction or end transaction operation on a database that was not opened by TPS.
6	The transaction record has an improper TRSTATE value.
7	The program attempted to execute a MIDTRANSACTION statement on a database that was not opened by TPS.
8	The program attempted to update an audited database while the program was not in transaction state.
9	The program attempted to abort a single transaction while the program was not in transaction state.
10	The program attempted to write a savepoint record while the program was not in transaction state.
11	The program attempted either to abort a single transaction or to write a savepoint record, but the DASDL option INDEPENDENTTRANS was not set for the database.

CLOSEERROR Subcategories

CLOSEERROR is major category 12. The most probable cause for a CLOSEERROR message is an error in closing the database.

Table A-4. CLOSEERROR Subcategories

Subcategory Number	Meaning
1	The database is not open.
2	An Accessroutines error occurred on one or more structures.

DATAERROR Subcategories

DATAERROR is major category 4. The most probable cause for a DATAERROR message is a null value in a required item.

Table A-5. DATAERROR Subcategories

Subcategory Number	Meaning
1	Either some required items are null, or the VERIFY condition failed.
2	The program attempted to create variable-format records with an invalid value for the record type.
3	The partition variable yielded an illegal file identifier.
4	READONLY items were changed and now cannot be stored.

DEADLOCK Subcategories

DEADLOCK is major category 3. The most probable cause for a DEADLOCK message is either a deadly embrace or deadlock condition.

Table A-6. DEADLOCK Subcategories

Subcategory Number	Meaning
1	A deadly embrace occurred when the program attempted to lock a record or while the program was waiting for a begin transaction operation. The system automatically frees all records for this program.
2	A timeout occurred when the program attempted to lock a record or while the program was waiting for a begin transaction operation.
3	The LOCKLIMIT was exceeded. Too many records in this structure are currently locked.

DUPLICATES Subcategories

DUPLICATES is major category 2. This category usually is reported when duplicates are not allowed.

Table A-7. DUPLICATES Subcategories

Subcategory Number	Meaning
1	The program attempted to store a duplicate key item in an index set in which duplicates are not allowed.
2	The program attempted to insert a duplicate key item in a subset or data set in which duplicates are not allowed.

FATALERROR Subcategories

FATALERROR is major category 19. The most probable cause for a FATALERROR message is a fatal error in the Accessroutines. No exception subcategories are returned for large systems.

INTEGRITYERROR Subcategories

INTEGRITYERROR is major category 20. The most probable cause for an INTEGRITYERROR message is an internal structure problem in the database.

Table A-8. INTEGRITYERROR Subcategories

Subcategory Number	Meaning
1	The key and the data do not match.
2	The program attempted to store a record in a missing automatic set or subset.
3	The program attempted to delete a record in a missing automatic set or subset.
4	An automatic set or subset is pointing to a deleted record.
5	The program attempted to allocate more than 1000 rows in a file.

INTLIBERROR Subcategories

INTLIBERROR is major category 21. This category relates only to exceptions and errors generated using the interpretive interface.

Table A-9. INTLIBERROR Subcategories

Subcategory Number	Meaning
1	The first parameter of the entry point DBVERB or DBDATA is not a valid language extension verb.
2	The specified string parameter is not a valid keyword.
3	The specified structure is not the correct structure type.
4	The structure or link name is not a known database identifier.
5	The program gave conflicting or insufficient qualification for the structure or link name.
6	The item name is not a database identifier.
7	The item type does not correspond to the requested data type.
8	The specified item is not in the specified structure.
9	The program gave an invalid index for an OCCURS item.
10	The program gave an incorrect number of subscripts for an OCCURS item.
11	Either the specified item is not in the current variable record type, or the program specified an incorrect record-type syntax.
12	The program placed a negative value in an unsigned item.
13	The program attempted a get operation for this item and found that the value was null.

continued

Table A-9. INTLIBERROR Subcategories (cont.)

Subcategory Number	Meaning
14	The string parameter in a putstring operation is not the same length as the item.
15	The structure or link name is invalid.
16	The item name is invalid.
17	A selection expression contains an invalid literal.
18	A right parenthesis is missing in the selection expression.
19	A relational operator is expected in the selection expression.
20	In a selection expression, either a relational operator specified a null value for a required item, or the relational operator was neither EQU (equal to) nor GEQ (greater than or equal to).
21	Either the selection expression did not end correctly, or some unknown error occurred.
22	A numeric argument to a put operation is invalid for the item.
23	This operation is not valid for global data.
24	In a selection expression, an item and a literal have either a mismatched length or type.
25	More than 47 relations appear in a selection expression.
26	No valid set spans the embedded data set.
27	A software failure occurred in a selection expression tree execution.
28	The program attempted to execute an invalid request while the database was closed.
29	The program attempted to execute a store or delete operation while the program was not in transaction state.
30	An index set was left undefined by a data set condition search.
31	The parameter is too short to hold the response.
32	The value of a number item does not fit into a real parameter.
33	The LIMITTYPE parameter is invalid.
34	The LIMITVALUE parameter is invalid.
35	The FINDLIMIT value was exceeded.

INUSE Subcategories

INUSE is major category 14. The most probable cause for an INUSE messages is a problem with a master record that has embedded records.

Table A-10. INUSE Subcategories

Subcategory Number	Meaning
1	The program attempted to delete a record without setting its reference and embedded set to null.
2	The program attempted to delete a record when the count item was not zero.
3	The program attempted to delete a record that has a nonnull embedded structure.

IOERROR Subcategories

IOERROR is major category 9. The most probable cause for an IOERROR message is an input or output error.

Table A-11. IOERROR Subcategories

Subcategory Number	Meaning
1	The unit is not ready.
2	A read parity error occurred.
4	A descriptor error occurred.
5	The Accessroutines encountered a NO SPACE condition. This type of exception can occur under the following conditions: • A DMSII data file is contained on a multipack family. • One of the packs of the multipack family is re-created. • A program attempted a DMSII operation involving a record on the newly re-created pack.
6	The program attempted to read past the end of the file.
7	The program attempted to read or write past the end of the last row.
8	The I/O was canceled.
11	A pack write-lockout occurred.
12	A multiplexor or controller error occurred.
13	The pack drive failed.
14	A pack sector format error occurred.
15	A storage unit timeout occurred.

continued

Table A-11. IOERROR Subcategories (cont.)

Subcategory Number	Meaning
16	A row was locked out because of a previous write error.
17	The checksum operation failed.
99	An error was generated by an unknown cause.
256	The ADDRESSCHECK operation failed.

KEYCHANGED Subcategories

KEYCHANGED is major category 6. The most probable cause for a KEYCHANGED message is a change to a key item when no duplicates are allowed.

Table A-12. KEYCHANGED Subcategories

Subcategory Number	Meaning
1	The program attempted to change the key item of a set or automatic subset that does not allow duplicate keys.

LIMITERROR Subcategories

LIMITERROR is major category 10. The most probable cause for a LIMITERROR message is a file reaching its maximum physical limit.

Table A-13. LIMITERROR Subcategories

Subcategory Number	Meaning
1	The program attempted to store too many records in a structure.
2	The program attempted to make too many counted links refer to one record. The count list in the DASDL source file was exceeded.
3	The program attempted to put too many entries into a set or subset.
5	The program attempted to open too many partitions.

continued

Table A-13. LIMITERROR Subcategories (cont.)

Subcategory Number	Meaning
6	The program referenced a partitioned structure in a DMUTILITY INITIALIZE statement.
7	There are too many pointers for this structure.

NORECORD Subcategories

NORECORD is major category 13. The most probable cause for a NORECORD message is an invalid current path.

Table A-14. NORECORD Subcategories

Subcategory Number	Meaning
1	The current path of the record to be inserted is not valid.
2	The current path of the record to be deleted (removed) is not valid.
3	The current path containing the link is not valid.
4	The current path of the record to be linked to is not valid.
5	The current path of the master record is not valid.
6	The current path of the record to be found is not valid.

NOTFOUND Subcategories

NOTFOUND is major category 1. A NOTFOUND message is most likely to be issued when no record can be found that meets the given criteria.

Table A-15. NOTFOUND Subcategories

Subcategory Number	Meaning
1	No record satisfies the condition.
2	The program attempted to execute a next operation when the pointer was at the last entry in the structure.
3	The program attempted to execute a prior operation when the pointer was at the first entry in the structure.
4	The manual subset key does not correspond with the key in the data record.
5	No current record could be found.
6	The current path is undefined.
7	Either the master record pointer is undefined, or the embedded structure has not been created.
8	The link is undefined.
9	The link does not agree with the data record.
10	The link does not exist in the current record. (The program specified the wrong type of variable-format record.)
11	A select-text error occurred.
12	An invalid index resulted from an OCCURS clause.

NOTLOCKED Subcategories

NOTLOCKED is major category 5. A NOTLOCKED message is most likely to be issued when a store operation is not preceded by a create or modify operation.

Table A-16. NOTLOCKED Subcategories

Subcategory Number	Meaning
1	The program attempted to execute a store operation without first performing a create, lock and modify, or store operation.
2	The program attempted to modify an unlocked record.

OPENERERROR Subcategories

OPENERERROR is major category 11. The most probable cause for an OPENERERROR message is an error in opening the database.

Table A-17. OPENERERROR Subcategories

Subcategory Number	Meaning
1	The Accessroutines is not present on disk.
2	An I/O error occurred in the Accessroutines code file.
4	DMUTILITY tried to initialize when the database was in use.
5	The run-time description of the database does not match the compile-time description.
6	An Accessroutines error occurred on one or more structures.
7	The description files for the application program and the DMSUPPORT library were compiled with different timestamps.
9	The program attempted to open RECOVERY, but RECOVERY was not necessary.
10	The program discontinued when the restart data set was opened.
11	A read error occurred on segment 0 of the restart data set.
12	The program discontinued when initializing the restart data set to empty.
13	The program discontinued when changing the names of the restart data set.
14	The restart data set is not present.
15	The program attempted to open RECOVERY, but RECOVERY was already in progress.
16	A required structure was not invoked.
17	An invoked structure was not found.
18	The program attempted to open RECONSTRUCTION when RECONSTRUCTION was already in progress.
19	The reconstruction was unsuccessful.
20	The DBS stack overflowed.
23	The program attempted to initialize one or more partitions when the partitions already existed.
24	The program discontinued because a file attribute error occurred when the program attempted to open a file.
25	The program attempted to open a logical database that did not exist.
26	The program received a security error when attempting to execute an open operation.

continued

Table A-17. OPENERROR Subcategories (cont.)

Subcategory Number	Meaning
27	The Accessroutines code file is not the same as the code file used by the database. If the Accessroutines is copied while the database is open, any subsequent program that opens the database also receives this error until the database is closed. Either specify the correct Accessroutines title in the DASDL source file, or remove the incorrect copy from the usercode and family of the program that opened the database.
28	The system was unable to read row 0 of the restart data set. Run the RECONSTRUCT program.
29	The program was compiled with a description file that has a higher update level than the description file used to compile the DMSUPPORT library.
31	An I/O error occurred in the control file.
32	The program attempted to execute an unknown type of open operation.
33	The database was open when a halt/load occurred.
34	The audit trail could not be initialized.
35	There is insufficient memory for the DBS stack.
36	The Accessroutines code file is unacceptable.
37	An open error occurred in the control file.
38	The database software does not match the format for the structure.
39	The control file is locked by an exclusive function.
40	The application program must be recompiled on a more recent release.
41	The program and the DMSUPPORT library are compiled with different description files.
44	The DBS is not visible from the subsystem for the program.
45	This option is not permitted while TPS is active.
46	This option is not permitted until the TPS RECOVERY program is complete.
47	The initiation of the Accessroutines failed.
48	The application program discontinued before the Accessroutines was frozen.
49	The control file is not present on disk.
50	The program received an I/O error while reading block 0 of the control file.
56	The Accessroutines code file does not support audit.

continued

Table A-17. OPENERERROR Subcategories (cont.)

Subcategory Number	Meaning
57	The Accessroutines code file does not support partitions.
58	The Accessroutines code file does not support local buffering.
59	The program attempted to open a database that is already in use by an exclusive or single update user.
60	The program attempted to either open a database exclusively or make a database a single update database when the database is already in use by another program.
61	The program attempted to open a Semantic Information Manager (SIM) database, but SIM has not been implemented on the system.
62	The program attempted to update a SIM database that can be accessed only for inquiry purposes.
63	The program attempted to update a DMSII database that can be accessed only for inquiry purposes.
64	The program attempted to library-equate a new DMSUPPORT library, but either there is an existing DMSUPPORT library or the old DMSUPPORT library is still being accessed.
65	The usercode under which the database control file is stored is not a valid usercode for the system. Either make the usercode a valid system usercode, or move the control file to an existing valid system usercode.
66	The DMSUPPORT library is not resident.
67	The DMSUPPORT library needs to be recompiled.
68	The Absolute Block Serial Number (ABSN) is approaching its maximum allowable value and must be reset. (For information on resetting the ABSN, refer to the DMSII Utilities Operations Guide.)
69	A rebuild or rollback procedure is in progress.

READONLY Subcategories

READONLY is major category 8. The most probable cause for a READONLY message is an attempt to alter a record when the database is open for inquiry only.

Table A-18. READONLY Subcategories

Subcategory Number	Meaning
1	The program attempted to change the database while in the inquiry access mode.
2	The program attempted an insert or remove operation on a READONLY record.
3	The program attempted a create operation for a new partition while in inquiry mode.

SECURITYERROR Subcategories

SECURITYERROR is major category 17. The most probable cause for a SECURITYERROR message is a violation of system security.

Table A-19. SECURITYERROR Subcategories

Subcategory Number	Meaning
1	Verb security: either the program or the user is not permitted to use this function.

SYSTEMERROR Subcategories

SYSTEMERROR is major category 7. The most probable cause for a SYSTEMERROR message is an invalid structure number.

Table A-20. SYSTEMERROR Subcategories

Subcategory Number	Meaning
1	The program attempted a find first, next, prior, or last operation on an embedded standard or compact data set.
2	A generate operation is not allowed on this type of set.
3	The system attempted to read beyond the end of the file.
4	An Accessroutines fault occurred during a DMSFREE operation.
5	An invalid calling sequence or other sequence of events occurred.

continued

Table A-20. SYSTEMERROR Subcategories (cont.)

Subcategory Number	Meaning
6	A fatal software error occurred. A fatal software error is produced in the following situations: • Internal system locks are used inconsistently. • Linkage between blocks of data is inconsistent. • Information read from mass storage device is inconsistent with information requested.
7	A fatal integrity error occurred. A fatal integrity error is produced in the following situations: • An address of 0 (zero) is encountered. • A block address is not on a block boundary. • A record address is not on a record boundary. • The value of a key item in a set or subset does not match the value in a data set. • A set or an automatic subset points to a deleted record. • A record does not have an entry in a set or an automatic subset when it should have such an entry.
8	A fatal I/O error occurred. A fatal I/O error occurs in the following situations: • The audit file cannot be opened. • Recovery information cannot be read or written. • The Access routines encountered I/O errors when attempting to allocate or deallocate file available space.
9	A fatal resource error occurred. A fatal resource error is encountered in the following situations: • The main memory is exhausted. • There is not enough space in the restart data set.
10	A fatal unknown error occurred. This error occurs when abort recovery terminates on an error.
11	A fatal operator discontinue database error occurred. This error occurs in the following situations: • A program attempted to open a database just after the operator discontinued the database. • The database is being terminated due to some fatal error.
12	A software error occurred in the control file handler.

continued

Table A-20. SYSTEMERROR Subcategories (cont.)

Subcategory Number	Meaning
13	The partitions control in the control file are not initialized.
14	The Accessroutines code file is compiled without this structure type.

VERSIONERROR Subcategories

VERSIONERROR is major category 18. The most probable cause for a VERSIONERROR message is a program-to-database mismatch.

Table A-21. VERSIONERROR Subcategories

Subcategory Number	Meaning
1	The link is unknown or was deleted.
2	The program attempted to reference a structure that has been deleted or changed since the program was compiled.

Appendix B

Sample Programs Using the Interpretive Interface

The ALGOL, COBOL74, and FORTRAN77 application programs shown in this appendix use the DMSII interpretive interface to create, store, retrieve, delete, and re-create various data items of differing types.

All the programs use the same database. The DASDL description of the database is presented first, followed by the programs.

DASDL Database for Interpretive Interface Sample Programs

The following DASDL code creates EMPJOB, the database that is accessed by all the sample application programs provided in this appendix.

EMPJOB contains a restart data set, a global record, and three data sets: EMP, JOB, and ADR.

- EMP has an embedded data set (EMP-ED), three sets (EMP-DATE-SET, EMP-NO-SET and EMP-LNAME-SET), and an automatic subset (EMP-OVER10-SUBSET).
- JOB is a variable format data set with two record types. JOB has a set (JOB-CODE-SET), an automatic subset (JOB-MNG-SUBSET), and an embedded, manual subset (JOB-EMP-SUBSET).
- ADR has one set, ADR-ZIP-SET.

For details on DASDL, read the DMSII DASDL Reference Manual. For database design techniques, read the DMSII Technical Overview.

RSTART RESTART DATA SET
 (ALPHAID ALPHA(7));

***** GLOBAL RECORD *****

FISCAL-YEAR "NEEDS UPDATING EVERY YEAR" NUMBER (6);
 POP-EMP POPULATION (50) OF EMP;
 DIV-NUMBER NUMBER (5);

***** EMP *****

EMP DATA SET "MASTER EMPLOYEE FILE"
 POPULATION = 50

```
(
EMP-NO                NUMBER      (4); %ACTUAL EMP NUMBER
EMP-NAME              GROUP
  \ (
  \ EMP-LNAME          ALPHA      (10);
  \ EMP-FNAME          ALPHA      (10);
  \ );
EMP-SSN               NUMBER      (9);
EMP-DATE-HIRED "YYMMDD" NUMBER      (6);
EMP-SEC-CLEAR "T=YES"  BOOLEAN
EMP-SALARY "MONTHLY"  NUMBER      (6,2);
```

\ %***** EMP-ED *****

EMP-ED DATA SET "EDUCATIONAL RECORDS FOR EACH EMPLOYEE"

```
\ POPULATION = 10
\ (
\ ED-DATE "YYMMDD"    NUMBER      (6);
\ ED-INST-NAME        ALPHA      (30);
\ ED-INST-CODE        NUMBER      (4); %UNIQUE TO SCHOOL
\ ED-COURSE           ALPHA      (10);
\ ED-GPA              NUMBER      (3,2); %GRADE PT AVR
\ );
ED-DATE-SET           SET OF EMP-ED KEY ED-DATE I-S;
EMP-JOB-REF           REFERENCE TO JOB COUNTED;
);
EMP-NO-SET            SET OF EMP KEY EMP-NO I-S;
EMP-LNAME-SET         SET OF EMP
                      KEY EMP-LNAME DUPLICATES I-S;
EMP-OVER10-SUBSET "VESTED EMPLOYEES" SUBSET OF EMP
\ WHERE (EMP-DATE-HIRED < 700000)
\ BIT VECTOR;
```

***** JOB *****

JOB DATA SET "REC TYPE 1 IS CLERICAL, REC TYPE 2 IS NON-CLERICAL"
 POPULATION = 10

```
(
JOB-COUNT             COUNT      (10);
JOB-TYPE              TYPE       (2);
```

DASDL Database for Interpretive Interface Sample Programs (cont.)

```

JOB-CODE          NUMBER      (4);
JOB-TITLE         ALPHA       (30);
JOB-EMP-SUBSET    SUBSET OF EMP KEY EMP-NO;
),
  1:
  (
    TYPE-SPEED          NUMBER      (2);
    SHORTHAND-SPEED     NUMBER      (2);
    CRTUSE-REQD         BOOLEAN     ;
  ),
  2:
  (
    SPANISH-REQD        BOOLEAN     ;
    DRIVING-REQD        BOOLEAN     ;
    JOB-COMMENT         ALPHA       (50);
  );
JOB-CODE-SET      SET OF JOB KEY JOB-CODE I-S;
JOB-MNG-SUBSET "MANAGERS" SUBSET OF JOB WHERE
  (JOB-CODE > 4999);

```

***** ADR *****

```

ADR DATA SET "MASTER ADDRESS FILE"
POPULATION = 50
(
  ADR-EMP-NO          NUMBER      (4);
  ADR-INFO            GROUP
  \ (
  \   ADR-STREET-NO    ALPHA       (30);
  \   ADR-ZIP          NUMBER      (5);
  \ );
);
ADR-ZIP-SET          SET OF ADR KEY ADR-ZIP I-S DUPLICATES;

```

ALGOL Application Program Using the Interpretive Interface

```

BEGIN
FILE REMOTEFILE
    (KIND=REMOTE
    ,UNITS=CHARACTERS
    ,MAXRECSIZE=1920
    ,MINRECSIZE=1
    ,FILETYPE=3
    );
$$ INCLUDE DMISYM="DATABASE/DMINTERPRETER" 20100000-20199999

%*****%
%* This program uses the ALGOL declaration of library entry points    %*
%* from the DATABASE/DMINTERPRETER symbolic file rather than        %*
%* explicitly declaring each entry point.                             %*
%*****%

%*****%
%*                               LOCAL DECLARATIONS                    %*
%*****%

REAL
    EXPR_1;
DOUBLE
    EXPR_2;
STRING
    DATA_SET_NAME
    ,FIRST_1
    ,PRIOR_1
    ,NEXT_1
    ,P_AUDIT
    ,OPEN_TYPE
    ,BEGIN_1
    ,CONDITION_1
    ,SPACE_1
    ,DATA_BUFFER
    ,DATA_REQUEST
    ,ITEM_NAME
    ,EXPR_3
    ,EXCEPTIONMSG;
BOOLEAN
    RSLT
    ,EXPR_4;

```

ALGOL Application Program Using the Interpretive Interface (cont.)

```
%*****%
%*          INTERNAL PROCEDURE TO HANDLE EXCEPTIONS          *%
%*****%

PROCEDURE EXCEPTIONHANDLER;
BEGIN
  STRING
    EXCEPTIONMSG;
  BOOLEAN
    STATUSRSLT;

  STATUSRSLT:=DBSTATUS;
  DBEXCEPTIONNAME(STATUSRSLT,EXCEPTIONMSG);
  WRITE(REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
  DBEXCEPTIONTEXT(STATUSRSLT,EXCEPTIONMSG);
  WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
  WRITE(REMOTEFILE,<"A60">,EXCEPTIONMSG);
END OF EXCEPTIONHANDLER;

%*****%
%*          BODY OF MAIN PROGRAM          *%
%*****%

  DMI.TITLE:="DMINTERPRETER/EMPJOB";
%*****%
  DATA_SET_NAME:="EMP";
  FIRST_1:="FIRST";
  PRIOR_1:="PRIOR";
  NEXT_1:="NEXT";
  P_AUDIT:="AUDIT";
  OPEN_TYPE:="UPDATE";
  BEGIN_1:="BEGINNING";
  CONDITION_1:="EMP-NO=11";
  SPACE_1:=" ";

%*****%
%*          OPENING THE DATABASE          *%
%*****%

  RSLT:=DBOPEN(OPEN_TYPE);
  IF RSLT
    THEN EXCEPTIONHANDLER;
  WRITE(REMOTEFILE,<"SUCCESSFUL OPEN OF EMPJOB">);
```

ALGOL Application Program Using the Interpretive Interface (cont.)

```

%*****%
%*                                     DELETING A RECORD                               *%
%*****%

    RSLT:=DBBEGINTRANSACTION(P_AUDIT);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    RSLT:=DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    CONDITION_1:="EMP-NO=12";
    RSLT:=DBDELETE(FIRST_1,DATA_SET_NAME,CONDITION_1);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    RSLT:=DBENDTRANSACTION(P_AUDIT);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    WRITE(REMOTEFILE, <"SUCCESSFUL DELETE OF RECORD">);

%*****%
%*                                     CREATING A RECORD                               *%
%*****%

    RSLT:=DBCREATE(DATA_SET_NAME,SPACE_1);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    WRITE(REMOTEFILE, <"SUCCESSFUL CREATE OF RECORD">);

%*****%
%*                                     FILLING IN A RECORD                               *%
%*****%
%
%      USING DBPUTREAL ENTRY POINT
%
    ITEM_NAME:="EMP-NO";
    EXPR_1:=11;
    RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
    IF RSLT
        THEN EXCEPTIONHANDLER;

    ITEM_NAME:="EMP-SALARY";
    EXPR_1:=4500.95;
    RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME, 2 * EXPR_1);
    IF RSLT
        THEN EXCEPTIONHANDLER;

```

```

%
%      USING DBPUTDOUBLE ENTRY POINT
%
ITEM_NAME:="EMP-SSN";
EXPR_2:=573230911;
RSLT:=DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );
IF RSLT
    THEN EXCEPTIONHANDLER;

%
%      USING DBDATA ENTRY POINT
%
DATA_REQUEST:="PUT REAL EMP EMP-DATE-HIRED";
DATA_BUFFER:="811223";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

DATA_REQUEST:="PUT STRING EMP EMP-LNAME";
DATA_BUFFER:="STACK";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

%
%      USING DBPUTSTRING ENTRY POINT
%
ITEM_NAME:="EMP-FNAME";
EXPR_3:="CATHY";
RSLT:=DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "LINA");
IF RSLT
    THEN EXCEPTIONHANDLER;

%
%      USING DBPUTBOOLEAN
%
ITEM_NAME:="EMP-SEC-CLEAR";
EXPR_4:=TRUE;
RSLT:=DBPUTBOOLEAN(DATA_SET_NAME,ITEM_NAME,NOT EXPR_4);
IF RSLT
    THEN EXCEPTIONHANDLER;

```

```

%*****%
%*                RECORD SETUP IN USER WORK AREA                %*
%*****%
%*                %*                %*                %*                %*                %*                %*
%* EMP      EMP      EMP      EMP      EMP      EMP      EMP      %*
%* NO      LNAME    FNAME    SSN      DATE-HIRED  SEC-CLEAR  SALARY  %*
%*-----%
%*21  STACK  CATHYLINA 573230900  811223      FALSE      9001.9 %*
%*****%

```

```

%*****%
%*                STORING A RECORD                                %*
%*****%

```

```

RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBSTORE(DATA_SET_NAME);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RECORD">);

```

```

%*****%
%*                RE-CREATING A RECORD                            %*
%*****%

```

```

RSLT:=DBRECREATE(DATA_SET_NAME,SPACE_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL RECREATE OF RECORD">);

ITEM_NAME:="EMP-NO";
EXPR_1:=12;
RSLT:=DBPUTREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

ITEM_NAME:="EMP-SSN";
EXPR_2:=549927328;
RSLT:=DBPUTDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2 - 11 );
IF RSLT
    THEN EXCEPTIONHANDLER;

```

```

DATA_REQUEST:="PUT REAL EMP EMP-DATE-HIRED";
DATA_BUFFER:="811228";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

DATA_REQUEST:="PUT STRING EMP EMP-LNAME";
DATA_BUFFER:="WOLVERTON";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;

ITEM_NAME:="EMP-FNAME";
EXPR_3:="LAURA";
RSLT:=DBPUTSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3 CAT "JANE" );
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBBEGINTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBSTORE(DATA_SET_NAME);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBENDTRANSACTION(P_AUDIT);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL STORE OF RE-CREATED RECORD">);

%*****%
%*          SETUP OF RE-CREATED RECORD IN USER WORK AREA          %*
%*****%
%*          %*
%* EMP      EMP      EMP      EMP      EMP      EMP      EMP      %*
%* NO      LNAME     FNAME     SSN     DATE-HIRED  SEC-CLEAR  SALARY %*
%*-----%*
%* 12 WOLVERTON LAURAJANE 549927317 811228      FALSE      9001.9 %*
%*****%

%*****%
%*          SETTING DATA SET TO THE BEGINNING                    %*
%*****%

RSLT:=DBSET(DATA_SET_NAME,BEGIN_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL DBSET OF DATA SET">);

```

```
%*****%
%*                FINDING A RECORD                *%
%*****%
```

```
RSLT:=DBFIND(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL FIND OF RECORD">);
```

```
%*****%
%*                LOCKING A RECORD                *%
%*****%
```

```
RSLT:=DBLOCK(FIRST_1,DATA_SET_NAME,CONDITION_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL LOCK OF RECORD">);
```

```
%*****%
%*                VERIFYING ITEMS IN THE RECORD    *%
%*****%
```

```
%
%                USING DBGETREAL ENTRY POINT
%
ITEM_NAME:="EMP-NO";
RSLT:=DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-NO=",I4>,EXPR_1);

ITEM_NAME:="EMP_SALARY";
RSLT:=DBGETREAL(DATA_SET_NAME,ITEM_NAME,EXPR_1);
IF RSLT
    THEN
WRITE(REMOTEFILE, <"EMP-SALARY=",F9.2>,EXPR_1);

%
%                USING DBGETDOUBLE ENTRY POINT
%
ITEM_NAME:="EMP-SSN";
RSLT:=DBGETDOUBLE(DATA_SET_NAME,ITEM_NAME,EXPR_2);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SSN=",I9>,EXPR_2);
```

```

%
%      USING DBDATA ENTRY POINT
%
DATA_REQUEST:="GET REAL EMP EMP-DATE-HIRED";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-DATE-HIRED=",A6>,DATA_BUFFER);

DATA_REQUEST:="GET REAL EMP EMP-SALARY";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SALARY=",A6>,DATA_BUFFER);

DATA_REQUEST:="GET STRING EMP EMP-LNAME";
RSLT:=DBDATA(DATA_REQUEST,DATA_BUFFER);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-LNAME=",A10>,DATA_BUFFER);

%
%      USING DBGETSTRING ENTRY POINT
%
ITEM_NAME:="EMP-FNAME";
RSLT:=DBGETSTRING(DATA_SET_NAME,ITEM_NAME,EXPR_3);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-FNAME=",A10>,EXPR_3);

%
%      USING DBGETBOOLEAN ENTRY POINT
%
ITEM_NAME:="EMP-SEC-CLEAR";
RSLT:=DBGETBOOLEAN(DATA_SET_NAME,ITEM_NAME,EXPR_4);
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE, <"EMP-SEC-CLEAR=",L5>,EXPR_4);

%*****%
%*                                     **%
%*****%

RSLT:=DBFREE(DATA_SET_NAME);
IF RSLT
    THEN EXCEPTIONHANDLER;

WRITE(REMOTEFILE, <"SUCCESSFUL FREEING OF RECORD">);

```

```

%*****%
%*                                     CAUSING AN EXCEPTION                               *%
%*****%
WRITE(REMOTEFILE, <"CAUSING AN EXCEPTION">);

RSLT:=DBSET(DATA_SET_NAME,BEGIN_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

RSLT:=DBFIND(PRIOR_1,DATA_SET_NAME,SPACE_1);
IF RSLT
    THEN EXCEPTIONHANDLER;

%
%*****%
%*                                     USING DBVERB                                   *%
%*****%
%
%       USING DBVERB ON A SET STATEMENT
%
RSLT:=DBVERB("SET EMP TO BEGINNING");
IF RSLT
    THEN EXCEPTIONHANDLER
WRITE(REMOTEFILE,<"SUCCESSFUL SET EMP TO BEGINNING">);

%
%       USING DBVERB ON A SEARCH STATEMENT
%
RSLT:=DBVERB("FIND FIRST EMP AT EMP-NO=11");
IF RSLT
    THEN EXCEPTIONHANDLER;
WRITE(REMOTEFILE,<"SUCCESSFUL FIND FIRST EMP AT EMP-NO=11">);

%
%       USING DBVERB ON A STATUS STATEMENT
%
RSLT:=DBVERB("SET EMP TO BEGINNING");
IF RSLT
    THEN EXCEPTIONHANDLER;
DBVERB("FIND PRIOR EMP ");
RSLT:=DBVERB("DMSTATUS"); % TO FIND OUT IF THE FIND WAS OK
DBEXCEPTIONNAME(RSLT,EXCEPTIONMSG);
WRITE (REMOTEFILE,<"EXCEPTION-NAME=",A14>,EXCEPTIONMSG);
DBEXCEPTIONTEXT(RSLT,EXCEPTIONMSG);
WRITE(REMOTEFILE,<"EXCEPTION-TEXT=">);
WRITE(REMOTEFILE,<A60>,EXCEPTIONMSG);
WRITE(REMOTEFILE,<"SUCCESSFUL STATUS STATEMENT">);

```

```
%*****%
%*               CLOSING THE DATABASE               **%
%*****%

      RSLT:=DBCLOSE;
      IF RSLT
        THEN EXCEPTIONHANDLER

      WRITE(REMOTEFILE, <"SUCCESSFUL CLOSE OF EMPJOB">);

%*****%
%*****%
END OF DMIEXAMPLE.
```

COBOL74 Application Program Using the Interpretive Interface

```

*****
IDENTIFICATION DIVISION.
*****
*PROGRAM-ID.
*AUTHOR.
*INSTALLATION.
*DATE-WRITTEN.
*DATE-COMPILED.
*SECURITY.      PUBLIC.
*REMARKS.
*THIS PROGRAM USES ENTRY POINTS OF THE DMINTERPRETER LIBRARY
*COMPILED FOR THE EMPJOB SAMPLE DATABASE.
*****
ENVIRONMENT DIVISION.
*****
CONFIGURATION SECTION.
*=====
SOURCE-COMPUTER. A 9.
OBJECT-COMPUTER. A 9.
*=====
INPUT-OUTPUT SECTION.
*=====
FILE-CONTROL.
    SELECT REMOTEFILE ASSIGN TO REMOTE.
*****
DATA DIVISION.
*****
FILE SECTION.
FD  REMOTEFILE
    RECORD CONTAINS 80 CHARACTERS
    VALUE OF TITLE IS "REMOTEFILE".
    01  REMOTE-REC          PIC X(80).
*****
WORKING-STORAGE SECTION.
*DMINTERPRETER CODE FILE.
*-----
    01  DMI-NAME             PIC X(24) VALUE "DMINTERPRETER/EMPJOB".
*DMI VOCABULARY.
*-----
    01  DATA-SET-NAME       PIC X(17) VALUE IS "EMP".
    01  ITEM-NAME            PIC X(17) VALUE IS SPACES.
    01  FIRST-1              PIC X(5) VALUE "FIRST".
    01  PRIOR-1              PIC X(5) VALUE "PRIOR".
    01  NEXT-1               PIC X(4) VALUE "NEXT".
    01  P-AUDIT              PIC X(5) VALUE "AUDIT".
    01  N-AUDIT              PIC X(8) VALUE "NOAUDIT".
    01  OPEN-TYPE            PIC X(6) VALUE "UPDATE".
    01  BEGIN-1              PIC X(9) VALUE "BEGINNING".

```

COBOL74 Application Program Using the Interpretive Interface (cont.)

```

01 CONDITION-1      PIC X(9) VALUE "EMP-NO=11".
01 SPACE-1          PIC X(1) VALUE " ".
01 DATA-BUFFER     PIC X(10) VALUE IS SPACES.
01 DATA-REQUEST    PIC X(40) VALUE IS SPACES.
01 VAR-1            PIC X(10) VALUE IS SPACES.

```

```

*
* EXCEPTION AREA
*

```

```

01 RESULT           PIC 9(1) COMP VALUE IS 0.
01 EXCEPTION-NAME   PIC X(14).
01 EXCEPTION-MSG    PIC X(80).
01 EXCEPTION-WORD   PIC 9(12) COMP.

```

***** EXCEPTION-WORD FIELDS*****

```

* 02 STRUCTURE      PIC 9(4) COMP.
* 02 CATEGORY       PIC 9(3) COMP.
* 02 SUB-CATEGORY   PIC 9(4) COMP.
* 02 FLAG           PIC 9(1) COMP.
*

```

```

PROCEDURE DIVISION.
MAIN-SECTION.

```

```

*****
*                                *
*                                *
*                                *
*                                *
*                                *

```

***** LINKING TO DMI LIBRARY *****

```

CHANGE ATTRIBUTE TITLE OF "DMINTERPRETER"
TO DMI-NAME.

```

```

OPEN OUTPUT REMOTEFILE.

```

```

*****
*                                *
*                                *
*                                *
*                                *
*                                *

```

```

CALL "DBOPEN OF DMINTERPRETER"
  USING OPEN-TYPE
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

```

COBOL74 Application Program Using the Interpretive Interface (cont.)

```

*****
                        DELETING A RECORD                               *
*****

```

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBDELETE OF DMINTERPRETER"
    USING FIRST-1, DATA-SET-NAME, CONDITION-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
MOVE "EMP-NO=12" TO CONDITION-1.
CALL "DBDELETE OF DMINTERPRETER"
    USING FIRST-1, DATA-SET-NAME, CONDITION-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBENDTRANSACTION OF DMINTERPRETER"
    USING P-AUDIT
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

```
*****
*                                CREATING A RECORD                                *
*****
```

```
CALL "DBCREATE OF DMINTERPRETER"
    USING DATA-SET-NAME, SPACE-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

* FILLING IN A RECORD *

```
*  
*  
*      USING DBDATA ENTRY POINT  
*  
  
MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.  
MOVE "11" TO DATA-BUFFER.  
CALL "DBDATA OF DMINTERPRETER"  
    USING DATA-REQUEST, DATA-BUFFER  
    GIVING RESULT.  
IF RESULT = 1  
    PERFORM EXCEPTION-HANDLER.
```

```

MOVE "PUT REAL EMP EMP-SALARY" TO DATA-REQUEST.
MOVE "9001.90" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
MOVE "573230911" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
MOVE "811223" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
MOVE "STACK" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

*
*           USING DBPUTDISPLAY ENTRY POINT
*

MOVE "EMP-FNAME" TO ITEM-NAME.
MOVE "CATHY" TO VAR-1.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

*
*           USING DBPUTBOOLEAN
*

MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
MOVE "TRUE" TO VAR-1.
CALL "DBPUTBOOLEAN OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

```

```
*****
*          RECORD SETUP IN USER WORK AREA          *
*****
* EMP    EMP    EMP    EMP    EMP    EMP    EMP    *
* NO    LNAME  FNAME  SSN    DATE-HIRED  SEC-CLEAR  SALARY  *
*-----*
*  11   STACK  CATHY  573230911  811223      TRUE    9001.90 *
*****
```

```
*****
*          STORING A RECORD                        *
*****
```

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
  USING P-AUDIT
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
CALL "DBSTORE OF DMINTERPRETER"
  USING DATA-SET-NAME
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
CALL "DBENDTRANSACTION OF DMINTERPRETER"
  USING P-AUDIT
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
*****
*          RE-CREATING A RECORD                    *
*****
```

```
CALL "DBRECREATE OF DMINTERPRETER"
  USING DATA-SET-NAME, SPACE-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-NO" TO DATA-REQUEST.
MOVE "12" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST, DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "PUT REAL EMP EMP-SSN" TO DATA-REQUEST.
MOVE "549927328" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
MOVE "811228" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "PUT STRING EMP EMP-LNAME" TO DATA-REQUEST.
MOVE "WOLVERTON" TO DATA-BUFFER.
CALL "DBDATA OF DMINTERPRETER"
    USING DATA-REQUEST,DATA-BUFFER
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

MOVE "EMP-FNAME" TO ITEM-NAME.
MOVE "LAURA" TO VAR-1.
CALL "DBPUTDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
```

```
*****
*          SETUP OF RE-CREATED RECORD IN USER WORK AREA          *
*****
* EMP      EMP      EMP      EMP      EMP      EMP      EMP      *
* NO      LNAME     FNAME     SSN      DATE-HIRED  SEC-CLEAR  SALARY *
*-----*
* 12  WOLVERTON  LAURA  549927328  811228      TRUE      9001.90 *
*****
```

```
CALL "DBBEGINTRANSACTION OF DMINTERPRETER"
  USING P-AUDIT
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
CALL "DBSTORE OF DMINTERPRETER"
  USING DATA-SET-NAME
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
CALL "DBENDTRANSACTION OF DMINTERPRETER"
  USING P-AUDIT
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
*****
*          SETTING DATA SET TO THE BEGINNING                    *
*****
```

```
CALL "DBSET OF DMINTERPRETER"
  USING DATA-SET-NAME, BEGIN-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
*****
*          FINDING A RECORD                                       *
*****
```

```
CALL "DBFIND OF DMINTERPRETER"
  USING FIRST-1, DATA-SET-NAME, CONDITION-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
*****
*                               LOCKING A RECORD                               *
*****
```

```
CALL "DBLOCK OF DMINTERPRETER"
  USING FIRST-1, DATA-SET-NAME, CONDITION-1
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
*****
*                               VERIFYING ITEMS IN RECORD                               *
*****
```

```
*
*                               USING DBDATA ENTRY POINT                               *
*
```

```
MOVE "GET REAL EMP EMP-NO" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-SALARY" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-SSN" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET REAL EMP EMP-DATE-HIRED" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```
MOVE "GET STRING EMP EMP-LNAME" TO DATA-REQUEST.
CALL "DBDATA OF DMINTERPRETER"
  USING DATA-REQUEST,DATA-BUFFER
  GIVING RESULT.
IF RESULT = 1
  PERFORM EXCEPTION-HANDLER.
```

```

*
*      USING DBGETDISPLAY ENTRY POINT
*
MOVE "EMP-FNAME" TO ITEM-NAME.
CALL "DBGETDISPLAY OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

*
*      USING DBGETBOOLEAN
*
MOVE "EMP-SEC-CLEAR" TO ITEM-NAME.
CALL "DBGETBOOLEAN OF DMINTERPRETER"
    USING DATA-SET-NAME,ITEM-NAME,VAR-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

*****
*                                     *
*      FREEING THE RECORD                                     *
*****

CALL "DBFREE OF DMINTERPRETER"
    USING DATA-SET-NAME
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

*****
*                                     *
*      CAUSING AN EXCEPTION                                   *
*****

CALL "DBFIND OF DMINTERPRETER"
    USING FIRST-1, DATA-SET-NAME, SPACE-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.
CALL "DBFIND OF DMINTERPRETER"
    USING PRIOR-1, DATA-SET-NAME, SPACE-1
    GIVING RESULT.
IF RESULT = 1
    PERFORM EXCEPTION-HANDLER.

```

```

*****
*                               *
*                               *
*****
*
*       USING DBVERB ON A SET STATEMENT
*
      MOVE "SET EMP TO BEGINNING" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
        USING DATA-REQUEST
        GIVING RESULT.
      IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.
*
*       USING DBVERB ON A SEARCH STATEMENT
*
      MOVE "FIND FIRST EMP AT EMP-NO=11" TO DATA-REQUEST.
      CALL "DBVERB OF DMINTERPRETER"
        USING DATA-REQUEST
        GIVING RESULT.
      IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

*****
*                               *
*                               *
*****

      CALL "DBCLOSE OF DMINTERPRETER"
        GIVING RESULT.
      IF RESULT = 1
        PERFORM EXCEPTION-HANDLER.

      CLOSE REMOTEFIL.
      STOP RUN.

EXCEPTION-HANDLER.

      CALL "DBSTATUS OF DMINTERPRETER"
        USING EXCEPTION-WORD.
      CALL "DBEXCEPTIONNAME OF DMINTERPRETER"
        USING EXCEPTION-WORD, EXCEPTION-NAME.
      MOVE EXCEPTION-NAME TO REMOTE-REC.
      WRITE REMOTE-REC.
      CALL "DBEXCEPTIONTEXT OF DMINTERPRETER"
        USING EXCEPTION-WORD, EXCEPTION-MSG.
      MOVE EXCEPTION-MSG TO REMOTE-REC.
      WRITE REMOTE-REC.

END-OF-PROGRAM.
EXIT PROGRAM.

```

FORTRAN77 Application Program Using the Interpretive Interface

```

*****
* This program uses the FORTRAN77 declaration of library entry points *
* from the DATABASE/DMINTERPRETER symbolic file rather than          *
* explicitly declaring each entry point.                               *
*****
*
$ RESET FREE
*
*****
*          FORTRAN77 LIBRARY AND ENTRY POINT DECLARATIONS          *
*****

      BLOCK GLOBALS
      LIBRARY DMI(TITLE="DMINTERPRETER/EMPJOB.")
      END

$ INCLUDE "DATABASE/DMINTERPRETER" 20200000-20249999

*****
*          SUBROUTINE EXCEPT          *
*****

      SUBROUTINE EXCEPT

$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

      CHARACTER      MSG      *70
      LOGICAL        RESULT

      RESULT = DBSTAT( )

      CALL DBEXCN(RESULT,MSG)
      WRITE(6,200) MSG

      CALL DBEXCT(RESULT,MSG)
      WRITE(6,200) MSG
200  FORMAT(1X,A)
      RETURN
      END

$ INCLUDE "DATABASE/DMINTERPRETER" 20290000-20299999

```

```
*****
*                                START OF MAIN PROGRAM                                *
*****
```

```
CHARACTER DSNAME      *4      /"EMP"/,
*      FIRST          *10     /"FIRST"/,
*      PRIOR           *10     /"PRIOR"/,
*      NEXT            *10     /"NEXT"/,
*      BEGIN           *9      /"BEGINNING"/,
*      AUDIT           *10     /"AUDIT"/,
*      UPDATE          *6      /"UPDATE"/,
*      SPACE           *1      /" "/,
*      COND            *9      /"EMP-NO=11"/,
*      DATREQ          *40     ,
*      DATBUF          *22     ,
*      ITMNAM          *13     ,
*      MSG             *70     ,
*      EXPR3           *22
```

```
LOGICAL  RSLT,
*        EXPR4
```

```
DOUBLE PRECISION  EXPR2
```

```
REAL  EXPR1
```

```
*****
*                                OPENING THE DATABASE                                *
*****
```

```
RSLT = DBOPEN(UPDATE)
IF (RSLT) CALL EXCEPT
```

```
*****
*                                DELETING A RECORD                                *
*****
```

```
RSLT = DBBTR(AUDIT)
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBDEL(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT
```

```
RSLT = DBETR(AUDIT)
IF (RSLT) CALL EXCEPT
```

```

*****
*                                CREATING A RECORD                                *
*****

      RSLT = DBCR(DSNAME,SPACE)
      IF (RSLT) CALL EXCEPT
*****
*                                FILLING IN A RECORD                                *
*****

*
*      USING DBPUTREAL ENTRY POINT
*
      ITMNAM = "EMP-NO"
      EXPR1 = 11
      RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-SALARY"
      EXPR1 = 4500.95
      RSLT = DBPUTR(DSNAME,ITMNAM, 2 * EXPR1)
      IF (RSLT) CALL EXCEPT

*
*      USING DBPUTDOUBLE ENTRY POINT
*
      ITMNAM = "EMP-SSN"
      EXPR2 = 573230911
      RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 11)
      IF (RSLT) CALL EXCEPT

*
*      USING DBDATA ENTRY POINT
*
      DATREQ = "PUT REAL EMP EMP-DATE-HIRED"
      DATBUF = "811223"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

      DATREQ = "PUT CHARACTER EMP EMP-LNAME"
      DATBUF = "STACK"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

*
*      USING DBPUTCHARACTER ENTRY POINT
*
      ITMNAM = "EMP-FNAME"
      EXPR3 = "CATHY"
      RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LINA")
      IF (RSLT) CALL EXCEPT

```

```

*
*      USING DBPUTLOGICAL ENTRY POINT
*
      ITMNAM = "EMP-SEC-CLEAR"
      EXPR4 = .TRUE.
      RSLT = DBPUTL(DSNAME,ITMNAM,.NOT. EXPR4)
      IF (RSLT) CALL EXCEPT

*****
*
*      RECORD SETUP IN USER WORK AREA
*
*****
*
*      EMP      EMP      EMP      EMP      EMP      EMP      EMP
*      NO      LNAME     FNAME     SSN      DATE-HIRED  SEC-CLEAR  SALARY
*-----*
*  22      STACK    CATHY    573230900  811223      FALSE      9001.9
*-----*
*****

*****
*
*      STORING A RECORD
*
*****

      RSLT = DBBTR(AUDIT)
      IF (RSLT) CALL EXCEPT

      RSLT = DBSTORE(DSNAME)
      IF (RSLT) CALL EXCEPT

      RSLT = DBETR(AUDIT)
      IF (RSLT) CALL EXCEPT

*****
*
*      RE-CREATING A RECORD
*
*****

      RSLT = DBRECR(DSNAME,SPACE)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-NO"
      EXPR1 = 12
      RSLT = DBPUTR(DSNAME,ITMNAM,EXPR1)
      IF (RSLT) CALL EXCEPT

      ITMNAM = "EMP-SSN"
      EXPR2 = 549273281
      RSLT = DBPUTD(DSNAME,ITMNAM, EXPR2 - 12)
      IF (RSLT) CALL EXCEPT

      DATREQ = "PUT STRING EMP EMP-LNAME"
      DATBUF = "BLACK"
      RSLT = DBDATA(DATREQ,DATBUF)
      IF (RSLT) CALL EXCEPT

```

```

ITMNAM = "EMP-FNAME"
EXPR3 = "SARAH"
RSLT = DBPUTC(DSNAME,ITMNAM,EXPR3(1:5) // "LEE")
IF (RSLT) CALL EXCEPT

```

```

RSLT = DBBTR(AUDIT)
IF (RSLT) CALL EXCEPT

```

```

RSLT = DBSTORE(DSNAME)
IF (RSLT) CALL EXCEPT

```

```

RSLT = DBETR(AUDIT)
IF (RSLT) CALL EXCEPT

```

```

*****
*                               SETTING DATA SET TO THE BEGINNING                               *
*****

```

```

RSLT = DBSET(DSNAME,BEGIN)
IF (RSLT) CALL EXCEPT

```

```

*****
*                               FINDING A RECORD                                           *
*****

```

```

RSLT = DBFIND(FIRST,DSNAME,COND).
IF (RSLT) CALL EXCEPT

```

```

*****
*                               LOCKING A RECORD                                           *
*****

```

```

RSLT = DBLOCK(FIRST,DSNAME,COND)
IF (RSLT) CALL EXCEPT

```

```

*****
*                               VERIFYING ITEMS IN A RECORD                               *
*****

```

```

*
*
*
*

```

```

    USING DBGETREAL ENTRY POINT

```

```

ITMNAM = "EMP-NO"
RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)
IF (RSLT) CALL EXCEPT

```

```

ITMNAM = "EMP-SALARY"
RSLT = DBGETR(DSNAME,ITMNAM,EXPR1)
IF (RSLT) CALL EXCEPT

```

```

*
*
*      USING DBGETDOUBLE ENTRY POINT
*
  ITMNAM = "EMP-SSN"
  RSLT = DBGETD(DSNAME,ITMNAM,EXPR2)
  IF (RSLT) CALL EXCEPT
*
*
*      USING DBDATA ENTRY POINT
*
  DATREQ = "GET REAL EMP EMP-DATE-HIRED"
  RSLT = DBDATA(DATREQ,DATBUF)
  IF (RSLT) CALL EXCEPT

  DATREQ = "GET CHARACTER EMP EMP-LNAME"
  RSLT = DBDATA(DATREQ,DATBUF)
  IF (RSLT) CALL EXCEPT
*
*
*      USING DBGETCHARACTER ENTRY POINT
*
  ITMNAM = "EMP-FNAME"
  RSLT = DBGETC(DSNAME,ITMNAM,EXPR3)
  IF (RSLT) CALL EXCEPT
*
*
*      USING DBGETLOGICAL ENTRY POINT
*
  ITMNAM = "EMP-SEC-CLEAR"
  RSLT = DBGETL(DSNAME,ITMNAM,EXPR4)
  IF (RSLT) CALL EXCEPT

*****
*                               FREEING THE RECORD                               *
*****

  RSLT = DBFREE(DSNAME)
  IF (RSLT) CALL EXCEPT

*****
*                               CAUSING AN EXCEPTION                               *
*****

  RSLT = DBSET(DSNAME,BEGIN)
  IF (RSLT) CALL EXCEPT

  RSLT = DBFIND(PRIOR,DSNAME,SPACE)
  IF (RSLT) CALL EXCEPT

```

```

*****
*                                     *
*                               USING DBVERB                               *
*                                     *
*****
*
*       USING DBVERB ON A SET STATEMENT
*
      RSLT = DBVERB("SET EMP TO BEGINNING")
      IF (RSLT) CALL EXCEPT
*
*       USING DBVERB ON A SEARCH STATEMENT
*
      RSLT = DBVERB("FIND FIRST EMP AT EMP-NO=11")
      IF (RSLT) CALL EXCEPT
*
*       USING DBVERB ON A STATUS STATEMENT
*
      RSLT = DBVERB("SET EMP TO BEGINNING")
      IF (RSLT) CALL EXCEPT
      RSLT = DBVERB("FIND PRIOR EMP")
      RSLT = DBVERB("DMSTATUS")
      CALL DBEXCN(RSLT,MSG)
      CALL DBEXCT(RSLT,MSG)

*****
*                                     *
*                               CLOSING THE DATABASE                       *
*                                     *
*****

      RSLT = DBCLOSE( )
      IF (RSLT) CALL EXCEPT

      END

```


Appendix C

Sample Programs Using Language Extensions

The ALGOL, COBOL74 and RPG application programs shown in this appendix use DMSII language extensions.

Notes: Consult the COBOL85 Reference Manual, Vol. 2, for COBOL85 programming examples.

The DMSII Technical Overview contains other COBOL74 programming examples using language extensions.

DASDL Database for the ALGOL Language Extension Program

The following DASDL code was compiled for the database \$UNIV. While this DASDL code illustrates many of the features of the DMSII language extensions, it is not intended to illustrate good database design techniques.

\$UNIV contains four disjoint standard data sets: UNIV-COURSES, UNIV-PERSONNEL, MSF, and ADR.

- UNIV-COURSES has two embedded data sets (BOOKS and STUDENTS) and one set (UNIV-C-SET). BOOKS has one subset (BOK). STUDENTS has one spanning set (STUDSET).
- UNIV-PERSONNEL has two sets (SS-U-P, U-P-SET) and an automatic subset (FREEPAY).
- MSF has an embedded data set (QUARTER), a set (MSFSET), an automatic subset (SEXSET), and two manual subsets (SMART, DMUTILITY). QUARTER has an embedded, variable-format data set (CORSES) and a set (QSET). CORSES has three possible formats (fixed, fixed and part 1, or fixed and part 2.)
- ADR has two sets (SAD, SSAD) and three subsets (STUAD, FACAD, and ADMAD).

FREEPAY, SEXSET, SMART, and DMUTILITY are bit vectors.

For details on DASDL, read the DMSII DASDL Reference Manual. For database design techniques, read the DMSII Technical Overview.

```

$ SET LIST SINGLE STORE TEST
UNIV-COURSES DATA SET "MAIN FILE"
(
  CRS-NAME          GROUP
  (
    DEPARTMENT      ALPHA (2);
    LEVEL           NUMBER (3);
    CRS-NO          NUMBER (4);
  ) REQUIRED;
  NOPROF            NUMBER (2);
  CNTOFCRS          COUNT (300);
  DAYS-OF-WEEK      FIELD
  (
    MON             BOOLEAN;
    TUES            BOOLEAN;
    WEDS            BOOLEAN;
    THURS           BOOLEAN;
    FRI             BOOLEAN;
    SAT             BOOLEAN;
  );
  BUILDING          NUMBER (3);
  ROOM              ALPHA (2) NULL IS "NO";
  COURSENAME        ALPHA (24);
  FLAG-BITS         FIELD (12);
  HOURSCRDY         NUMBER (4);
  CLASS-SIZE        NUMBER (2);
  PROFESSOR IS IN UNIV-PERSONNEL COUNTED OCCURS 3 TIMES;
  BOOKS             UNORDERED DATA SET
  (
    LC              NUMBER (9);
    TITLE           ALPHA (60) NULL IS BLANKS;
    AUTHOR          ALPHA (30);
  ) BUFFERS         = 1 + 1 PER USER,
    AREAS           = 10,
    AREASIZE        = 500,
    POPULATION      = 5,
    BLOCKSIZE       = 5;
  BOK SUBSET OF BOOKS UNORDERED LIST DATA LC;
  STUDENTS          DATA SET
  (
    LAST-NAME       ALPHA (15) REQUIRED;
    FIRST-NAME      ALPHA (10) REQUIRED;
    MSTR IS IN MSFSET;
    MSLK IS KEY OF MSFSET
  ) POPULATION = 300;
  STUDSET SET OF STUDENTS KEY IS (LAST-NAME ASCENDING,
    FIRST-NAME) I-S DUPLICATES
    LOADFACTOR      = 75
    TABLESIZE      = 12
    AREAS            = 100
  )
% END RECORD DESCRIPTION OF UNIV-COURSES DATA SET

```

ALGOL Application Program Using Language Extensions (cont.)

```
POPULATION = 1000
VERIFY (HOURSCRTD GTR 0 AND CLASS-SIZE LEQ 60) AND NOPROF NEQ 0;
UNIV-C-SET SET OF UNIV-COURSES KEY IS CRS-NAME DESCENDING I-S
NO DUPLICATES;
%
UNIV-PERSONNEL DATA SET
POPULATION = 997
(
  USC-COUNT          COUNT (100);
  NAME               GROUP
  (
    LASTNAME          ALPHA (15);
    FIRSTNAME         ALPHA (10)
  ) REQUIRED;
  SEX                BOOLEAN;
  AGE                NUMBER (2) NULL IS HIGH-VALUE;
  SSNUM              NUMBER (9) REQUIRED UNIQUE;
  DPT                ALPHA (4);
  RANK               ALPHA (1);
  SALARY             NUMBER (5,2) INITIALVALUE IS LOW-VALUE;
  COURSES IS IN UNIV-COURSES COUNTED OCCURS 8 TIMES;
  ADDRESS IS IN ADR VERIFY ON PHON;
  SUPR IS IN UNIV-PERSONNEL WITH NO PROTECTION
);
SS-U-P SET OF UNIV-PERSONNEL KEY IS SSNUM I-S NO DUPLICATES;
U-P-SET SET OF UNIV-PERSONNEL KEY IS NAME INDEX SEQUENTIAL DUPLICATES;
%
MSF DATA SET
(
  SSNO               NUMBER (9) REQUIRED UNIQUE;
  NONAM              NUMBER (1);
  LNAME              ALPHA (30);
  MNAME              ALPHA (30) OCCURS 9 TIMES;
  FNAME              ALPHA (30);
  CAMPUS-ADDRESS     GROUP
  (
    DORM              ALPHA (6);
    ROOM              NUMBER (4);
    POBOX             NUMBER (4);
    PHONE             NUMBER (7)
  );
  ND                 NUMBER (2);
  DEGREE             ALPHA (4) OCCURS 6 TIMES;
  TOTHR              NUMBER (3);
  TOTQP              REAL (3);
  GRADE-POINT-AVG    NUMBER (3,2);
  MJR                NUMBER (3);
  AMJR              ALPHA (18);
  SSEX              BOOLEAN;
  SAGE               NUMBER (2);
  HOME-ADDRESS IS IN ADR VERIFY ON PHON;
  QSET SET OF QUARTER KEY QTR NO DUPLICATES O-L;
```

```

QUARTER DATA SET
(
    QTR                      ALPHA (4) REQUIRED;
    QTTRHRS                  NUMBER (2);
    QTRQP                    NUMBER (2);
    CSEET SET OF CORSES KEY TYPECOURSE DUPLICATES;
    POPCORSSES POPULATION (100000) OF CORSES;
    CORSES DATA SET
    POPULATION = 4, BLOCKSIZE = 30 WORDS
    (
        TYPECOURSE          NUMBER (1) REQUIRED;
        CORSETYPE           TYPE (2);
    )
    % END FIXED PORTION OF CORSES
    1:
    (
        GRADE                ALPHA (2) REQUIRED;
        CRS IS IN UNIV-COURSES COUNTED;
    )
    2:
    (
        YR                   NUMBER (2);
        Q                    NUMBER (2);
        GCRS IS IN UNIV-COURSES COUNTED;
        GGD                  ALPHA (2);
        TITLE-OF-PAPER       ALPHA (30);
        PPGRD                ALPHA (2);
    )
    )
    % END OF QUARTER DATA SET
)
% END OF RECORD DESCRIPTION OF MSF DATA SET
LOCK TO MODIFY DETAILS % PHYSICAL OPTIONS OF MSF
BLOCKSIZE = 6
POPULATION = 5000
AREAS = 100
KIND = DISK;
MSFSET SET OF MSF KEY IS SSNO I-S NO DUPLICATES;
%
ADR DATA SET
(
    FACULTY-STUDENT         NUMBER (1);
    SNO                     NUMBER (9) REQUIRED;
    NUMLNS                  NUMBER (1);
    ADLN                    ALPHA (30) OCCURS 9 TIMES;
    ZIPC                    NUMBER (5) REQUIRED;
    PHON                    NUMBER (12)
);
POPADMAD POPULATION (14) OF ADMAD;
SAD SET OF ADR KEY IS ZIPC I-S DUPLICATES FIRST;
SSAD SET OF ADR KEY IS SNO I-S DUPLICATES LAST;
STUAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 1)

```

KEY IS (ZIPC, SNO) I-S DUPLICATES;
FACAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 2)
KEY IS (ZIPC, SNO) I-R DUPLICATES MODULUS = 97;
ADMAD SUBSET OF ADR WHERE (FACULTY-STUDENT EQL 3)
KEY IS (ZIPC, SNO) I-R DUPLICATES;
FREEPAY SUBSET OF UNIV-PERSONNEL WHERE (SALARY LSS 0 OR SALARY EQL 0)
BIT VECTOR;
SEXSET SUBSET OF MSF WHERE (SAGE GEQ 21 AND NOT SSEX) BIT VECTOR;
SMART SUBSET OF MSF BIT VECTOR;
DMUTILITY SUBSET OF MSF BIT VECTOR;

ALGOL Application Program Using Language Extensions

The following ALGOL program example invokes only a portion of the database. Although the program uses all the ALGOL DMSII statements, it does not use all the available options. Moreover, even though the program shows several different ways of handling data management exceptions, it does not illustrate consistently good programming techniques.

```
$ SET LIST LISTDB
BEGIN

FILE CARD (KIND=READER);

FILE MONITORDMS (KIND=PRINTER);

EBCDIC ARRAY CARDREC [0:79];
DEFINE
    CTYPE      = CARDREC [0] #,
    CSSNO      = CARDREC [1] #,
    CGRDPTAVG  = CARDREC [10] #,
    CSEX       = CARDREC [15] #,
    CAGE       = CARDREC [16] #,
    CQTR       = CARDREC [18] #,
    CTYPECOURSE = CARDREC [22] #,
    CGRADE     = CARDREC [23] #,
    CTITLEPAPER = CARDREC [25] #,
    CNAME      = CARDREC [55] #;
EBCDIC ARRAY MONITORREC [0:131];
DEFINE
    MONITOREXCEPTION = MONITORREC [0] #,
    MONITORSTATUSB   = MONITORREC [4] #,
    MONITORSTATUS    = MONITORREC [24] #,
    MONITORVERB      = MONITORREC [30] #,
    MONITORSET       = MONITORREC [50] #,
    MONITORSTRUCTURE = MONITORREC [67] #;
EBCDIC ARRAY MSG [0:79];
INTEGER
    I,
    SNO,
    SSNO,
    TOOMANYEXCEPTIONS;
REAL
    DMSTATUS,
    GRADEPOINTAVG;
```

```

LABEL
    BEGINPROGRAM,
    CREATEADR,
    CREATEMOREADR,
    BUILDMSF,
    READCARDLOOP,
    CREATMSF100,
    CREATEQUARTER200,
    CREATECOURSE300,
    LINKMSFTOADR400,
    DELETEADR500,
    GENERATESUBSET600,
    CHANGEMSNAME700,
    CHANGEGRADE800,
    EOJ,
    XIT;
DATABASE UNIV:MSF,ADR;
%
% LINES BEGINNING WITH AN * ARE PRODUCED BY THE $SET LISTDB OPTION.
%
*DATABASE TITLE: UNIV ON DISK
* GLOBAL DATA ITEMS:
*02 INTEGER POPADMAD: POPULATION OF ADMAD
*01 MSF: DATA SET (#11)
* INVOKED SETS:
*     MSFSET (#16, AUTOMATIC), KEY = SSNO
*     SEXSET (#24, AUTOMATIC)
*     SMART (#25, MANUAL)
*     DMUTILITY (#26, MANUAL)
* RECORD ITEMS:
*02 INTEGER SSNO: NUMBER(9)
*02 INTEGER NONAM: NUMBER(1)
*02 STRING LNAME: ALPHA(30)
*02 STRING MNAME[1:9]: ALPHA(30)
*02 STRING FNAME: ALPHA(30)
*02 STRING CAMPUS-ADDRESS: GROUP(14)
*03     STRING DORM: ALPHA(6)
*03     INTEGER ROOM: NUMBER(4)
*03     INTEGER POBOX: NUMBER(4)
*03     INTEGER PHONE: NUMBER(7)
*02 INTEGER ND: NUMBER(2)
*02 STRING DEGREE[1:6]: ALPHA(4)
*02 INTEGER TOTHR: NUMBER(3)
*02 INTEGER TOTQP: REAL(3)
*02 REAL GRADE-POINT-AVG: NUMBER(3,2)
*02 INTEGER MJR: NUMBER(3)
*02 STRING AMJR: ALPHA(18)
*02 BOOLEAN SSEX
*02 INTEGER SAGE: NUMBER(2)
*02 HOME-ADDRESS: LINK TO ADR

```

```

*02    QUARTER: DATA SET (#13)
*      INVOKED SETS:
*        QSET (#12, AUTOMATIC), KEY = QTR
*      RECORD ITEMS:
*03    INTEGER POPCOURSE: POPULATION OF COURSES
*03    STRING QTR: ALPHA(4)
*03    INTEGER QTTRHRS: NUMBER(2)
*03    INTEGER QTRQP: NUMBER(2)
*03    COURSES: DATA SET (#15)
*      INVOKED SETS:
*        CSEET (#14, AUTOMATIC), KEY = TYPECOURSE
*      RECORD ITEMS:
*04    INTEGER CORSETYPE: RECORD TYPE
*04    INTEGER TYPECOURSE: NUMBER(1)
*      ITEMS OF RECORD FORMAT #2:
*04    INTEGER YR: NUMBER(2)
*04    INTEGER Q: NUMBER(2)
*04    STRING GGD: ALPHA(2)
*04    STRING TITLE-OF-PAPER: ALPHA(30)
*04    STRING PPGRD: ALPHA(2)
*04    GCRS: LINK TO UNIV-COURSES
*      ITEMS OF RECORD FORMAT #1:
*04    STRING GRADE: ALPHA(2)
*04    CRS: LINK TO UNIV-COURSES
*01 ADR: DATA SET (#17)
*      INVOKED SETS:
*        SAD (#18, AUTOMATIC), KEY = ZIPC
*        SSAD (#19, AUTOMATIC), KEY = SNO
*        STUAD (#20, AUTOMATIC), KEY = (ZIPC, SNO)
*        FACAD (#21, AUTOMATIC), KEY = (ZIPC, SNO)
*        ADMAD (#22, AUTOMATIC), KEY = (ZIPC, SNO)
*      RECORD ITEMS:
*02    INTEGER FACULTY-STUDENT: NUMBER(1)
*02    INTEGER SNO: NUMBER(9)
*02    INTEGER NUMLNS: NUMBER(1)
*02    STRING ADLN[1:9]: ALPHA(30)
*02    INTEGER ZIPC: NUMBER(5)
*02    INTEGER PHON: NUMBER(12)
*      DESCRIPTION TIMESTAMP: 04/21/82 | 15:15:51

```

```

PROCEDURE STATUSBOOLEAN;
BEGIN
  TOOMANYEXCEPTIONS := TOOMANYEXCEPTIONS + 1;
  IF TOOMANYEXCEPTIONS > 10 THEN
    BEGIN
      REPLACE MSG BY TOOMANYEXCEPTIONS FOR * DIGITS,
        "IS TOO MANY EXCEPTIONS", 48"00";
      DISPLAY (MSG);
      GO TO XIT;
    END;
  REPLACE MONITOREXCEPTION BY "*" FOR 4;
  REPLACE MONITORSTRUCTURE BY DMSTATUS.DMSTRUCTURE FOR * DIGITS;
  CASE DMSTATUS.DMERROR OF
    BEGIN
      NOTFOUND      :REPLACE MONITORSTATUSB BY "NOTFOUND      " ;
      DUPLICATES    :REPLACE MONITORSTATUSB BY "DUPLICATE    " ;
      DEADLOCK      :REPLACE MONITORSTATUSB BY "DEADLOCK      " ;
      DATAERROR    :REPLACE MONITORSTATUSB BY "DATAERROR    " ;
      NOTLOCKED     :REPLACE MONITORSTATUSB BY "NOTLOCKED     " ;
      KEYCHANGED    :REPLACE MONITORSTATUSB BY "KEYCHANGED    " ;
      SYSTEMERROR   :REPLACE MONITORSTATUSB BY "SYSTEMERROR   " ;
      READONLY      :REPLACE MONITORSTATUSB BY "READONLY      " ;
      IOERROR       :REPLACE MONITORSTATUSB BY "IOERROR       " ;
      LIMITERROR    :REPLACE MONITORSTATUSB BY "LIMITERROR    " ;
      OPENERERROR   :REPLACE MONITORSTATUSB BY "OPENERERROR   " ;
      CLOSEERROR    :REPLACE MONITORSTATUSB BY "CLOSEERROR    " ;
      NORECORD      :REPLACE MONITORSTATUSB BY "NORECORD      " ;
      INUSE         :REPLACE MONITORSTATUSB BY "INUSE         " ;
      AUDITERROR    :REPLACE MONITORSTATUSB BY "AUDITERROR    " ;
      ABORT         :REPLACE MONITORSTATUSB BY "ABORT         " ;
      VERSIONERROR  :REPLACE MONITORSTATUSB BY "VERSIONERROR  " ;
    END;
  WRITE (MONITORDMS,22,MONITORREC);
  REPLACE MONITORREC BY " " FOR 132;
END STATUSBOOLEAN;

```

```

%
BEGINPROGRAM:
  REPLACE MONITORREC BY " " FOR 132;
  OPEN UPDATE UNIV;
  FIND UNIV;          % GLOBAL DATA
  GET UNIV(I:=POPADMAD);
  REPLACE MSG BY I FOR * DIGITS, 48"00";
  DISPLAY (MSG);
CREATEADR:
  CREATE ADR;
  PUT ADR(FACULTY-STUDENT:=1,
          ZIPC:=1,
          PHON:=1,
          ADLN[1]:="FIRST AVE");
  GET ADR(SNO:=SNO);
  PUT ADR(SNO:=(SNO:=SNO+1));
  STORE ADR:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO EOJ;
    END;
CREATEMOREADR:
  RECREATE ADR;
  PUT ADR(SNO:=(SNO:=SNO+1));
  STORE ADR:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO EOJ;
    END;
  IF SNO < 11 THEN GO TO CREATEMOREADR;
%
% BUILD MASTER STUDENT FILE (MSF)
%
BUILDMSF:
  CARD.OPEN := TRUE;
READCARDLOOP:
  READ (CARD,14,CARDREC)[EOJ];
  WRITE (MONITORDMS,14,CARDREC);
  IF CTYPE = "1" THEN GO TO CREATMSF100;
  IF CTYPE = "2" THEN GO TO CREATEQUARTER200;
  IF CTYPE = "3" THEN GO TO CREATECOURSE300;
  IF CTYPE = "4" THEN GO TO LINKMSFTOADR400;
  IF CTYPE = "5" THEN GO TO DELETEADR500;
  IF CTYPE = "6" THEN GO TO GENERATESUBSET600;
  IF CTYPE = "7" THEN GO TO CHANGEMSFFNAME700;
  IF CTYPE = "8" THEN GO TO CHANGEGRADE800;
  REPLACE MSG BY CTYPE FOR 1, " INVALID CARDTYPE", 48"00";
  DISPLAY (MSG);
  GO TO XIT;

```

```

CREATEMSF100:
  SSNO := INTEGER(CSSNO,9);
  IF SSNO < 1 OR SSNO > 10 THEN
    BEGIN
      REPLACE MONITORREC BY
        "CSSNO COLS 2-10 MUST BE BETWEEN 0 AND 11";
      WRITE (MONITORCMS,22,MONITORREC);
      REPLACE MONITORREC BY " " FOR 132;
      GO TO READCARDLOOP;
    END;
  CREATE MSF;
  PUT MSF(SSNO:=SSNO,
    GRADE-POINT-AVG:=
      (GRADEPOINTAVG:=(INTEGER(CGRDPTAVG,5)/100)),
    SSEX:=CSEX="M",
    SAGE:=INTEGER(CAGE,2),
    LNAME:=CNAME);
  STORE MSF:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO EOJ;
    END;
  IF GRADEPOINTAVG > 3.50 THEN
    INSERT MSF INTO SMART:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    STATUSBOOLEAN;
  GO TO READCARDLOOP;
CREATEQUARTER200:
  LOCK MSFSET AT SSNO = INTEGER(CSSNO,9):DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      IF DMSTATUS.DMERROR = NOTFOUND THEN
        BEGIN
          REPLACE MSG BY CSSNO FOR 9, "NOT IN MSF", 48"00";
          DISPLAY (MSG);
        END
      ELSE
        STATUSBOOLEAN;
      GO TO READCARDLOOP;
    END;
  CREATE QUARTER;
  PUT QUARTER(QTR:=CQTR);
  STORE QUARTER;
  GO TO READCARDLOOP;

```

```

CHANGEGRADE800:
  REPLACE MONITORVERB BY "LOCK MSFSET          ";
  LOCK MSFSET AT SSNO = INTEGER(CSSNO,9):DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO READCARDLOOP;
    END;
  GET MSF (GRADEPOINTAVG := GRADE-POINT-AVG);
  IF GRADEPOINTAVG GEQ 3.50 THEN
    BEGIN
      SET SMART TO MSF:DMSTATUS;
      IF BOOLEAN(DMSTATUS) THEN
        BEGIN
          STATUSBOOLEAN;
          GO TO READCARDLOOP;
        END;
      REPLACE MONITORVERB BY "REMOVE CURRENT          ";
      REMOVE CURRENT FROM SMART:DMSTATUS;
      IF BOOLEAN(DMSTATUS) THEN
        BEGIN
          STATUSBOOLEAN;
          GO TO READCARDLOOP;
        END;
      END;
    END;
  PUT MSF(GRADE-POINT-AVG:=
    (GRADEPOINTAVG:=(INTEGER(CGRDPTAVG,5)/100)));
  REPLACE MONITORVERB BY "STORE MSF          ";
  STORE MSF:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN STATUSBOOLEAN;
  IF GRADEPOINTAVG LSS 3.50 THEN GO TO READCARDLOOP;
  REPLACE MONITORVERB BY "SET SMART          ";
  SET CURRENT TO MSF:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO READCARDLOOP;
    END;
  %
  % IN PLACE OF THE "SET" AND "REMOVE" STATEMENTS, USE
  % "SET SMART TO MSF" AND "REMOVE MSF FROM SMART".
  %
  GO TO READCARDLOOP;
EOJ:
  IF TOOMANYEXCEPTIONS = 0 THEN
    BEGIN
      REPLACE MSG BY "NO DM EXCEPTION", 48"00";
      DISPLAY (MSG);
    END;
  CLOSE UNIV:DMSTATUS;
XIT:
END.

```

COBOL74 Application Program Using Language Extensions

The following sample program, SAILOLPROG, tracks sailboat races and updates a DMSII database by using features of the COMS direct-window interface. The program exemplifies the programming techniques used in writing transaction processors that allow synchronized recovery.

The program maintains the DMSII database SAILDB. The database contains three data sets. The data set RACE-CALENDAR contains one record for every race. The data set ENTRY contains one record for each boat entered in the race. A boat may have multiple records, depending on the number of races it enters. The data set RDS is the restart data set and has the required COMS fields. The DASDL option RDSSTORE is not set for the database.

The program includes the following COMS entities:

- A direct window called SAIL
- An agenda called SAILAGINOL
- Three trancodes:
 - CRERAC(MFI = 1)
 - ADDENT(MFI = 2)
 - DELENT(MFI = 3)

All entities must be defined to COMS to allow the program to run.

The program contains three transactions. Each transaction has a unique trancode and a unique module function index (MFI). The transaction with the trancode CRERAC creates a race entry in the database. The transaction with the trancode ADDENT adds a boat entry to a race. (The race must exist for the transaction to be completed.) The transaction with the trancode DELENT deletes a boat from a race.

All transactions in the program are two-phase transactions. In phase one, all records are locked. In phase two, the data is stored in the database and only the end transaction operation unlocks records. All transactions instruct COMS to audit the input message when the end transaction is executed.

The program also contains the statements required by the TADS system. The program must be compiled specifying the TADS option and the TADS diagnostic station. No changes must be made to the program if the TADS system is not present.

The program code contains comments to indicate what the program is doing at each step.

```

CREATECORSES300:
  LOCK MSFSET AT SSNO = INTEGER(CSSNO,9):DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      IF DMSTATUS.DMERROR = NOTFOUND THEN
        BEGIN
          REPLACE MSG BY CSSNO FOR 9, "NOT IN MSF", 48"00";
          DISPLAY (MSG);
        END
      ELSE
        STATUSBOOLEAN;
        GO TO READCARDLOOP;
      END;
    LOCK QSET AT QTR=CQTR;
    IF CTYPECOURSE = "1" THEN CREATE CORSES(1);
    IF CTYPECOURSE = "2" THEN CREATE CORSES(2);
    PUT CORSES(CTYPECOURSE:=INTEGER(CTYPECOURSE,1));
    IF CTYPECOURSE = "1" THEN
      PUT CORSES(GRADE:=CGRADE)
    ELSE
      PUT CORSES(GGD:=CGRADE);
    STORE CORSES:DMSTATUS;
    IF BOOLEAN(DMSTATUS) THEN STATUSBOOLEAN;
    GO TO READCARDLOOP;
LINKMSFTOADR400:
  LOCK MSFSET AT SSNO = INTEGER(CSSNO,9);
  LOCK SSAD AT SNO = INTEGER(CSSNO,9);
  ASSIGN ADR TO HOME-ADDRESS;
  GO TO READCARDLOOP;
DELETEADR500:
  REPLACE MONITORVERB BY "LOCK MSFSET          ";
  LOCK MSFSET AT SSNO = INTEGER(CSSNO,9):DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO XIT;
    END;
  REPLACE MONITORVERB BY "LOCK HOMELL          ";
  LOCK HOME-ADDRESS:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO XIT;
    END;
  REPLACE MONITORVERB BY "ASSIGN NULL          ";
  ASSIGN NULL TO HOME-ADDRESS:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      STATUSBOOLEAN;
      GO TO XIT;
    END;

```

```

REPLACE MONITORVERB BY "DELETE ADRL      ";
DELETE. ADR:DMSTATUS;
IF BOOLEAN(DMSTATUS) THEN
  BEGIN
    STATUSBOOLEAN;
    GO TO XIT;
  END;
GO TO READCARDLOOP;
GENERATESUBSET600:
  REPLACE MONITORVERB BY "GENERATE AND      ";
  GENERATE DMUTILITY = SMART AND SEXSET;
  FIND FIRST DMUTILITY:DMSTATUS;
  IF BOOLEAN(DMSTATUS) THEN
    BEGIN
      IF DMSTATUS.DMERROR = NOTFOUND THEN
        BEGIN
          REPLACE MSG BY "SUBSET EMPTY", 48"00";
          DISPLAY (MSG);
        END
      ELSE
        STATUSBOOLEAN;
        GO TO READCARDLOOP;
    END;
  GET MSF(MONITORREC:=LNAME);
  WRITE (MONITORDMS,22,MONITORREC);
  REPLACE MONITORREC BY " " FOR 132;
  REPLACE MONITORVERB BY "GENERATE NULL      ";
  GENERATE DMUTILITY = NULL;
  GO TO READCARDLOOP;
CHANGEMSNAME700:
  MODIFY MSFSET AT SSNO = INTEGER(CSSNO,9);
  GET MSF(MONITORREC:=LNAME);
  WRITE (MONITORDMS,22,MONITORREC);
  REPLACE MONITORREC BY " " FOR 132;
  REPLACE MONITORREC BY "NAME IN MSF WAS CHANGED TO ";
  WRITE (MONITORDMS,22,MONITORREC);
  REPLACE MONITORREC BY " " FOR 132;
  PUT MSF(LNAME:=CNAME);
  STORE MSF;
  REPLACE MONITORREC BY CNAME FOR 24;
  WRITE (MONITORDMS,22,MONITORREC);
  REPLACE MONITORREC BY " " FOR 132;
  GO TO READCARDLOOP;

```

```

010100$SET TADS(REMOTE RMT)
010200 IDENTIFICATION DIVISION.
010300 PROGRAM-ID. ONLINESAIL.
010400 ENVIRONMENT DIVISION.
010500 CONFIGURATION SECTION.
010600     SOURCE-COMPUTER. A-9.
010700     OBJECT-COMPUTER. A-9.
010800 INPUT-OUTPUT SECTION.
010900 FILE-CONTROL.
011000     SELECT RMT ASSIGN TO REMOTE ACCESS SEQUENTIAL.
011100 DATA DIVISION.
011200 FILE SECTION.
011300 FD RMT.
011400 01 REM-REC                                PIC X(72).
011500 DATA-BASE SECTION.
011600     DB SAILDB ALL.
011700 WORKING-STORAGE SECTION.
011800 77 SCRATCH                                PIC X(256).
011900
012000*****
012100* MESSAGE AREA                                *
012200*****
012300
012400 01 MSG-TEXT.
012500     03 MSG-TCODE                                PIC X(6).
012600     03 MSG-FILLER                                PIC X.
012700     03 MSG-CREATE-RACE.
012800         05 MSG-CR-ID                            PIC 9(6).
012900         05 MSG-CR-NAME                            PIC X(20).
013000         05 MSG-CR-DATE                            PIC X(6).
013100         05 MSG-CR-TIME                            PIC X(4).
013200         05 MSG-CR-LOCATION                          PIC X(20).
013300         05 MSG-CR-SPONSOR                          PIC X(10).
013350         05 FILLER                                PIC X(10).
013400     03 MSG-ADD-ENTRY REDEFINES MSG-CREATE-RACE.
013500         05 MSG-AE-RACE-ID                          PIC 9(6).
013600         05 MSG-AE-ID                              PIC X(6).
013700         05 MSG-AE-NAME                            PIC X(20).
013800         05 MSG-AE-RATING                          PIC 9(3).
013900         05 MSG-AE-OWNER                            PIC X(20).
014000         05 MSG-AE-CLUB                            PIC X(15).
014100         05 FILLER                                PIC X(6).
014200     03 MSG-DELETE-ENTRY REDEFINES MSG-CREATE-RACE.
014300         05 MSG-DE-RACE-ID                          PIC 9(6).
014400         05 MSG-DE-ID                              PIC X(6).
014500         05 FILLER                                PIC X(64).
014600     03 MSG-STATUS                                PIC X(30).
014700

```

```

014800 COMMUNICATION SECTION.
*****
* On Mark 3.7 COMS headers were implemented. This comment aids *
* you in comparing the declaration of COMS headers with the use *
* of binary CDs. Binary CDs will be deimplemented on the *
* Mark 3.9 release *
*
014900*-- CD COMS-IN USAGE BINARY; FOR INPUT; *
015000*-- SYMBOLIC QUEUE IS COMS-IN-PROGRAM; *
015100*-- SYMBOLIC SUB-QUEUE-1 IS COMS-IN-FUNCTION-INDEX; *
015200*-- SYMBOLIC SUB-QUEUE-2 IS COMS-IN-USERCODE; *
015300*-- SYMBOLIC SUB-QUEUE-3 IS COMS-IN-SECURITY-DESG; *
015400*-- MESSAGE DATE IS COMS-IN-DATE; *
015500*-- MESSAGE TIME IS COMS-IN-TIMESTAMP; *
015600*-- SYMBOLIC SOURCE IS COMS-IN-STATION; *
015700*-- TEXT LENGTH IS COMS-IN-TEXT-LENGTH; *
015800*-- END KEY IS COMS-IN-END-KEY; *
015900*-- STATUS KEY IS COMS-IN-STATUS-KEY; *
016000*-- MESSAGE COUNT IS COMS-IN-RST-LOC; *
016100*-- CONVERSATION AREA IS COMS-IN-CONV SIZE IS 60. *
016200*-- *
016300*-- CD COMS-OUT USAGE BINARY; FOR OUTPUT; *
016400*-- DESTINATION COUNT IS COMS-OUT-COUNT; *
016500*-- TEXT LENGTH IS COMS-OUT-TEXT-LENGTH; *
016600*-- STATUS KEY IS COMS-OUT-STATUS-KEY; *
016700*-- DESTINATION TABLE OCCURS 1 TIMES; *
016800*-- ERROR KEY IS COMS-OUT-DESTIN-ERROR; *
016900*-- SYMBOLIC DESTINATION IS COMS-OUT-DESTINATION. *
*
*****
017000
017002 INPUT HEADER COMS-IN;
017004 PROGRAMDESG IS COMS-IN-PROGRAM;
017006 FUNCTIONSTATUS IS COMS-IN-FUNCTION-STATUS;
017008 FUNCTIONINDEX IS COMS-IN-FUNCTION-INDEX;
017010 USERCODE IS COMS-IN-USERCODE;
017012 SECURITYDESG IS COMS-IN-SECURITY-DESG;
017014 TRANSPARENT IS COMS-IN-TRANSPARENT;
017016 VTFLAG IS COMS-IN-VT-FLAG;
017018 TIMESTAMP IS COMS-IN-TIMESTAMP;
017020 STATION IS COMS-IN-STATION;
017022 TEXTLENGTH IS COMS-IN-TEXT-LENGTH;
017024 STATUSVALUE IS COMS-IN-STATUS-KEY;
017026 MESSAGECOUNT IS COMS-IN-MSG-COUNT;
017028 RESTART IS COMS-IN-RST-LOC;
017030 AGENDA IS COMS-IN-AGENDA;
017032 SDFINFO IS COMS-IN-SDF-INFO.
017034
017036 OUTPUT HEADER COMS-OUT;
017038 DESTCOUNT IS COMS-OUT-COUNT;
017040 TEXTLENGTH IS COMS-OUT-TEXT-LENGTH;
017042 STATUSVALUE IS COMS-OUT-STATUS-KEY;

```

```

017044    TRANSPARENT                IS COMS-OUT-TRANSPARENT;
017046    VTFLAG                     IS COMS-OUT-VT-FLAG;
017048    CONFIRMFLAG                 IS COMS-OUT-CONFIRM-FLAG;
017050    CONFIRMKEY                   IS COMS-OUT-CONFIRM-KEY;
017052    DESTINATIONDESG             IS COMS-OUT-DESTINATION;
017054    NEXTINPUTAGENDA             IS COMS-OUT-NEXT-INPUT-AGENDA;
017056    SETNEXTINPUTAGENDA          IS COMS-OUT-SET-NEXT-INPUT-AGENDA;
017058    RETAINTRANSACTIONMODE       IS COMS-OUT-SAVE-TRANS-MODE;
017060    AGENDA                      IS COMS-OUT-AGENDA;
017062    SDFINFO                    IS COMS-OUT-SDF-INFO.
017100*****
017200    PROCEDURE DIVISION.
017300*****
017400    DECLARATIVES.
017500    DMERROR-SECTION SECTION.
017600        USE ON DMERROR.
017700    DMERROR-PARA.
017800
017900    END DECLARATIVES.
018000
018100    005-MAIN SECTION.
018200    005-MAIN-SN.
018500        CHANGE ATTRIBUTE LIBACCESS OF
018600            "DCILIBRARY" TO BYFUNCTION.
018700        CHANGE ATTRIBUTE FUNCTIONNAME OF
018800            "DCILIBRARY" TO "COMSSUPPORT".
018900        OPEN UPDATE SAILDB.
019000        IF DMSTATUS(DMERROR) CALL SYSTEM DMTERMINATE.
019100        ENABLE INPUT COMS-IN KEY "ONLINE".
019200        CREATE RDS.
019300*    MOVE COMS-IN-PROGRAM           TO RDS-PROG.
019400*    MOVE COMS-IN-RST-LOC           TO RDS-LOCATOR.
019420        PERFORM 007-PROCESS-COMS-INPUT
019440            UNTIL COMS-IN-STATUS-KEY = 99.
019500    005-MAIN-EXIT.
019510        PERFORM 910-CLOSEDOWN.
019520        STOP RUN.
019530
019540*****
019550    007-PROCESS-COMS-INPUT SECTION.
019560*****
019562*    Get the next message from COMS.  If it is 99, go to EOT.
019564*    Otherwise, make sure the message is valid before processing it.
019566
019570    007-PROCESS-CI-SN.
019600        MOVE SPACES TO MSG-TEXT.
019700        RECEIVE COMS-IN MESSAGE INTO MSG-TEXT.
019800        IF COMS-IN-STATUS-KEY NOT = 99
020000            PERFORM 920-CHECK-COMS-INPUT-ERRORS
020100            IF ( COMS-IN-STATUS-KEY = 0 OR 92) AND
020200                COMS-IN-FUNCTION-STATUS NOT < 0 THEN
020300                PERFORM 100-PROCESS-TRANSACTION.

```

```

020400 007-PROCESS-CI-EXIT.
020450     EXIT.
020500
021000*****
021100 100-PROCESS-TRANSACTION SECTION.
021200*****
021220* Since the transaction type is programmatically based on MFI,
021240* make sure the type is within range before doing the GO TO.
021260
021300 100-PROCESS-TRANS-SN.
021400     IF MSG-TCODE = "CRERAC"
021500         PERFORM 200-CREATE-RACE
021600     ELSE
021700     IF MSG-TCODE = "ADDENT"
021800         PERFORM 300-ADD-ENTRY
021900     ELSE
022000     IF MSG-TCODE = "DELENT"
022100         PERFORM 400-DELETE-ENTRY
022200     ELSE
022300         MOVE "INVALID TRANS CODE" TO MSG-STATUS
022320         PERFORM 900-SEND-MSG.
022340
022400 100-PROCESS-TRANS-EXIT.
022500     EXIT.
022600
022700*****
022800 200-CREATE-RACE SECTION.
022900*****
023000 200-CREATE-RACE-SN.
023020* Enter a new race record in the database.
023030* Since the transaction is done in online mode,
023040* save the restart data in the conversation area only.
023050* If there is an abort on BEGIN-TRANSACTION or on
023060* END-TRANSACTION, return to the RECEIVE statement.
023080
023100     CREATE RACE-CALENDAR.
023200     MOVE MSG-CR-NAME      TO RACE-NAME.
023300     MOVE MSG-CR-ID        TO RACE-ID.
023400     MOVE MSG-CR-DATE      TO RACE-DATE.
023500     MOVE MSG-CR-TIME      TO RACE-TIME.
023600     MOVE MSG-CR-LOCATION TO RACE-LOCATION.
023700     MOVE MSG-CR-SPONSOR  TO RACE-SPONSOR.
023800
023900*-- BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO-AUDIT RDS
023950     BEGIN-TRANSACTION COMS-IN              NO-AUDIT RDS
024000         ON EXCEPTION
024100         IF DMSTATUS(ABORT)
024200*             THEN MUST GET BACK TO RECEIVE STMT
024300             GO TO 200-CREATE-RACE-EXIT
024400         ELSE
024500             CALL SYSTEM DMTERMINATE.
024600*-- MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.

```

```

024700      STORE RACE-CALENDAR.
024800      IF DMSTATUS(DMERROR)
024900          MOVE "STORE ERROR" TO MSG-STATUS
025000      ELSE
025100          MOVE "RACE ADDED" TO MSG-STATUS.
025200      END-TRANSACTION COMS-OUT AUDIT RDS
025300      ON EXCEPTION
025400          IF DMSTATUS(ABORT)
025500              GO TO 200-CREATE-RACE-EXIT
025600      ELSE
025700          CALL SYSTEM DMTERMINATE.
025800      PERFORM 900-SEND-MSG.
025900 200-CREATE-RACE-EXIT.
026000      EXIT.
026100
026200*****
026300 300-ADD-ENTRY SECTION.
026400*****
026500 300-ADD-ENTRY-SN.
026520* Enter a boat in a race. The restart requirements are the same
026540* as for the CREATE-RACE section.
026560
026600      FIND RACE-SET AT RACE-ID = MSG-AE-RACE-ID
026700      ON EXCEPTION
026800          IF DMSTATUS(NOTFOUND)
026900              MOVE "RACE DOES NOT EXIST" TO MSG-STATUS
027000              PERFORM 900-SEND-MSG
027100              GO TO 300-ADD-ENTRY-EXIT
027200      ELSE
027300          CALL SYSTEM DMTERMINATE.
027400
027500      CREATE ENTRY.
027600      MOVE MSG-AE-NAME      TO ENTRY-BOAT-NAME.
027700      MOVE MSG-AE-ID        TO ENTRY-BOAT-ID.
027800      MOVE MSG-AE-RATING    TO ENTRY-BOAT-RATING.
027900      MOVE MSG-AE-OWNER     TO ENTRY-BOAT-OWNER.
028000      MOVE MSG-AE-CLUB      TO ENTRY-AFF-Y-CLUB.
028100      MOVE MSG-AE-RACE-ID   TO ENTRY-RACE-ID.
028200
028300*-- BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO-AUDIT RDS
028350      BEGIN-TRANSACTION COMS-IN          NO-AUDIT RDS
028400      ON EXCEPTION
028500          IF DMSTATUS(ABORT)
028600*          THEN MUST GET BACK TO RECEIVE STMT
028700          GO TO 300-ADD-ENTRY-EXIT
028800      ELSE
028900          CALL SYSTEM DMTERMINATE.
029000*-- MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.
029100      STORE ENTRY.
029200      IF DMSTATUS(DMERROR)
029300          MOVE "STORE ERROR" TO MSG-STATUS
029400      ELSE

```

```

029500      MOVE "BOAT ADDED" TO MSG-STATUS.
029600      END-TRANSACTION COMS-OUT AUDIT RDS
029700      ON EXCEPTION
029800      IF DMSTATUS(ABORT)
029900      GO TO 300-ADD-ENTRY-EXIT
030000      ELSE
030100      CALL SYSTEM DMTERMINATE.
030200      PERFORM 900-SEND-MSG.
030300 300-ADD-ENTRY-EXIT.
030400      EXIT.
030500
030600*****
030700 400-DELETE-ENTRY SECTION.
030800*****
030900 400-DELETE-ENTRY-SN.
030920* Delete a boat from a race. The restart requirements are the
030940* same as for the CREATE-RACE section.
030960
031000      LOCK ENTRY-RACE-SET AT
031100      ENTRY-RACE-ID = MSG-DE-RACE-ID AND
031200      ENTRY-BOAT-ID = MSG-DE-ID
031300      ON EXCEPTION
031400      IF DMSTATUS(NOTFOUND) THEN
031500      MOVE "BOAT ENTRY NOT FOUND" TO MSG-STATUS
031600      GO TO DE-SEND-MSG
031700      ELSE
031800      CALL SYSTEM DMTERMINATE.
031900
032000*-- BEGIN-TRANSACTION COMS-IN USING MSG-TEXT NO-AUDIT RDS
032050      BEGIN-TRANSACTION COMS-IN NO-AUDIT RDS
032100      ON EXCEPTION
032200      IF DMSTATUS(ABORT)
032300*      THEN MUST GET BACK TO RECEIVE STMT
032400      GO TO 400-DELETE-ENTRY-EXIT
032500      ELSE
032600      CALL SYSTEM DMTERMINATE.
032700*-- MOVE COMS-IN-RST-LOC TO RDS-LOCATOR.
032800      DELETE ENTRY.
032900      IF DMSTATUS(DMERROR)
033000      MOVE "FOUND BUT NOT DELETED" TO MSG-STATUS
033100      ELSE
033200      MOVE "BOAT DELETED" TO MSG-STATUS.
033300      END-TRANSACTION COMS-OUT AUDIT RDS
033400      ON EXCEPTION
033500      IF DMSTATUS(ABORT)
033600      GO TO 400-DELETE-ENTRY-EXIT
033700      ELSE
033800      CALL SYSTEM DMTERMINATE.
033900      DE-SEND-MSG.
034000      PERFORM 900-SEND-MSG.
034100 400-DELETE-ENTRY-EXIT.
034200      EXIT.

```

```

034300
034400*****
034500 900-SEND-MSG SECTION.
034600*****
034700 900-SEND-MSG-SN.
034720* Send the message back to the originating station. Do not specify
034740* an output agenda. Make sure to test the result of the SEND.
034760
034800      MOVE 1              TO COMS-OUT-COUNT.
034900*--  MOVE 0              TO COMS-OUT-DESTINATION(1).
034950      MOVE 0              TO COMS-OUT-DESTINATION.
035000      MOVE 0              TO COMS-OUT-STATUS-KEY.
035100      MOVE 106            TO COMS-OUT-TEXT-LENGTH.
035200      SEND COMS-OUT FROM MSG-TEXT.
035300      IF COMS-OUT-STATUS-KEY = 0 OR 92
035400          NEXT SENTENCE
035500      ELSE
035600          DISPLAY "ONLINE PROGRAM SEND ERROR: " COMS-OUT-STATUS-KEY.
035700 900-SEND-MSG-EXIT.
035800      EXIT.
035900*****
036000 910-CLOSEDOWN SECTION.
036100*****
036200 910-CLOSEDOWN-SN.
036220*      Close the database.
036240
036300*--  BEGIN-TRANSACTION NO-AUDIT RDS.
036400*--  RECREATE RDS.
036500*--  STORE RDS.
036600*--  END-TRANSACTION NO-AUDIT RDS.
036700      CLOSE SAILDB.
036800 910-CLOSEDOWN-EXIT.
036900      EXIT.
037000*****
037100 920-CHECK-COMS-INPUT-ERRORS SECTION.
037200*****
037300 920-CHECK-CIE-SN.
037320*      Check for COMS control messages.
037340
037400      IF ( COMS-IN-STATUS-KEY = 0 OR 92 OR 99 )
037500*      A good message, a recovery message, or an EOT notification.
037600          NEXT SENTENCE
037700      ELSE
037800          IF COMS-IN-STATUS-KEY = 93
037900              MOVE "MSG CAUSES ABORT, PLS DONT RETRY" TO MSG-STATUS
038000              PERFORM 900-SEND-MSG
038100          ELSE
038200*      It must be a 20, 100, 101, or 102, which means the
038300*      application is manipulating the dynamic attachment or
038400*      detachment of stations and received an error.
038500          MOVE "ERROR IN STA ATTACH/DETACHMENT" TO MSG-STATUS
038600          PERFORM 900-SEND-MSG.

```

```
038700
038800     IF COMS-IN-FUNCTION-STATUS < 0 THEN
038900*     This means that the application ID tied to a default input
039000*     agenda. MSG-TEXT probably does not contain a valid
039100*     transaction.
039100         MOVE "NEGATIVE FUNCTION CODE " TO MSG-STATUS
039200         PERFORM 900-SEND-MSG THRU 900-SEND-MSG-EXIT.
039300
039400 920-CHECK-CIE-EXIT.
039500     EXIT.
039600
```

DASDL Database for the RPG Language Extension Program

The following DASDL code creates the database used in the RPG sample application program. The audited database contains two data sets and two sets. The restart data set is RDS and has a set called RDSSET. The standard data set called EVENTS has a set called EDATES.

```
$ SET ZIP RPG
INITIALIZE;
DEFAULTS( KIND = PACK,
          PACKNAME = DISK % < ** CHANGE TO YOUR DATABASE PACK NAME
        );
OPTIONS( AUDIT STATISTICS ALLOWEDCORE = 5000);
AUDIT TRAIL( BLOCKSIZE = 900,
            AREASIZE = 60 BLOCKS,
            KIND = PACK,
            PACKNAME = DISK % < ** CHANGE TO YOUR AUDIT TRAIL PACK NAME
          );

RDS RESTART DATA SET
(
  RDS-ID      ALPHA(6) COMS-ID      % The program identification
              INITIALVALUE "ONLINE";
  RDSPRG      REAL COMS-PROGRAM;    % The program designator
  RDSLOC      REAL COMS-LOCATOR;    % The restart designator
  RDSFIL      ALPHA(12);            % The file identification
  RDSFLC      NUMBER(11);           % The file location
);

RDSSET SET OF RDS
  KEY (RDS-ID, RDSPRG)
  DUPLICATES;

EVENTS DATA SET
(
  E-CAP      NUMBER(6);             % Event capacity
  E-SOLD      NUMBER(6);             % Tickets sold so far
  E-DATE      NUMBER(6);             % Date of event
  E-NAME      ALPHA(20);             % Name of event
  E-PRCE      NUMBER(5,2);           % Price per ticket
)
  BLOCKSIZE = 100 RECORDS,
  AREASIZE = 20 BLOCKS
  POPULATION = 10000;

EDATES SET OF EVENTS
  KEY (E-DATE, E-NAME);
```

RPG Application Program Using Language Extensions

The following RPG application program updates the database for online ticket sales and adds and deletes events. The program initializes data, initializes COMS, and then receives and processes transactions until the status is *GO-EOT. Full-featured COMS is required to run this program. The program uses online synchronized recovery through COMS.

The following indicators and identifiers are used in the program:

Indicator	Meaning
06	Record-identifying indicator for restart data set RDS.
11	Record-identifying indicator for input header INHDR. An 11 indicates an "unknown" form type (which includes open and on notification).
12	Record-identifying indicator for input header INHDR. A 12 indicates that a HOME screen has been received.
13	Record-identifying indicator for input header INHDR. A 13 indicates that an ADD event screen has been received.
14	Record-identifying indicator for input header INHDR. A 14 indicates that a DELETE event screen has been received.
15	Record-identifying indicator for input header INHDR. A 15 indicates that a SELL tickets screen has been received.
19	Used to indicate unsuccessful linkage to CREDITLIB.
20	Compare indicators. Always temporary usage.
50	Credit indicator. ON means good credit; OFF means bad credit.
70	Indicator conditioning STORE with ADD into data set EVENTS.
71	Indicator conditioning STORE for MODIFY of data set EVENTS.
80	Set to ON when BYE screen is to be transmitted.
82	Set to ON to send SELL screen.
83	Set to ON to send ADD screen.
84	Set to ON to send DELETE screen.
85	Set to ON to send HOME screen.
90	Set to ON to initialize SEND screen.
91	Set to ON to send report of an event on SHOW screen.
92	Set to ON to send end-of-SHOW report.
94	Set to ON when ticket sale report is to be printed.
LR	End-of-file indicator for RECV (column 58). LR is used because the end-of-file indicator is set to ON when COMS returns a status of *GO-EOT. Setting the LR indicator to ON and exiting detail calculations causes the program to terminate.
Identifier	Meaning
CARD	See data structure REQUEST.
CARDEX	See data structure REQUEST.

Identifier	Meaning
CARDNO	See data structure REQUEST.
CUSADR	See data structure REQUEST.
CUSNAM	See data structure REQUEST.
DATE	See data structure REQUEST.
DUE	(8-digit, 2-decimal numeric). Internal field indicating the total amount due from a customer if a sale is made.
DUMMY	Dummy input file used as primary. Contains a single record only; data is not looked at.
EDATES	DMSII set of data set EVENTS. Key is E-DATE (date of event). Duplicate dates are allowed.
ENDTIK	Tag for end of subroutine TICKET. Used for early exit.
EVENT	See data structure REQUEST.
E-CAP	(5-digit numeric). Field of record from DMSII data set EVENTS. This field indicates the seating capacity for the requested event.
E-DATE	(6-digit numeric). Field of record from DMSII data set EVENTS. This field indicates the date on which the event will take place.
E-NAME	(20-byte alpha). Field of record from DMSII data set EVENTS. This field indicates the name of the event.
E-PRCE	(5-digit, 2-decimal numeric). Field of record from DMSII data set EVENTS. This field indicates the price per ticket for the event.
E-SOLD	(6-digit numeric). Field of record from DMSII data set EVENTS. This field indicates the number of tickets that have been sold so far for the event.
INHDR	The input header used for receiving input screens from the operator.
LINK	Tag used to loop until CREDITLIB is successfully linked to.
ODT	Operator Display Terminal. Used for DSPLYs.
OK	(2-byte alpha). Field used for accept portion of DSPLY.
OUTHDR	The output header used to send screens to operator.
QUANT	See data structure REQUEST.
QUIT	(1-byte alpha). The first byte received for every screen. On each screen, the operator specifies "quit" by putting a nonblank entry into the QUIT field.
REC	(150-byte alpha). Field used to capture the entire record of each RECV operation so that it can be audited at TRBEG. (If 3.7 DMSII is used, this field is unnecessary; DMSII performs the audit automatically.)
REQUEST	Data structure representing one ticket order (request).
RESTD	Restart Designator. The RESTART attribute of the input header INHDR.
TCOUNT	(5-digit numeric). Local field for the number of tickets being requested.
T-LEFT	(6-digit numeric). The number of tickets left for the current requested event.

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
00010H*
00015H*----- Control Specifications -----
00020H*
00025H* The "C" in column 15 causes the program to link to the COMS
00030H* that started the program. (This facility is generally used to
00035H* link to an experimental version of COMS.)
00040H*
00045H* If column 15 has a "D", the program links to the COMS that
00050H* started the program and can perform the DEBUG operation.
00055H*
00060H      C
00050D*
00100D*----- Database Specifications -----
00150D*
00200DTICKETDB                                U
00250F*
00300F*----- File Specifications -----
00350F*
00400FDUMMY  IP  F      80          DISK
00450A EXCLUSIVE 'FALSE'
00500FODT    D  F      80          ODT
00550F*
00600FPRINT  O  F     132          PRINTER
00650F*
00700FRDS    R                      DMS          A
00750F*
00800FEVENTS UD                    DMS          A
00850$$ PAGE
00900T*
00950T*----- Telecommunication Specifications -----
07000T*
07050T* Declare an input header as INHDR.
07100TINHDR  IHDR
07150A* The field PROGD captures the program designator for this
07200A* program. The designator must be saved in the restart data set.
07250A PROGRAMDESG                                PROGD
07300A* The field RESTD captures the restart locator for the header.
07350A* The locator must be saved in the restart data set.
07400A RESTART                                RESTD
07450T*
07500T* Declare an output header as OUTHDR.
07550TOUTHDR OHDR

```

```

.....1....*.....2....*.....3....*.....4....*.....5....*.....6....*.....7..
07600$$ PAGE
07650L*
07700L*----- Library Specifications -----
07750L*
07800L* The library function CREDITCHK sets the resulting indicator to
07850L* ON if the credit is OK, and sets it to OFF if it is not.
07900L* CREDITCHK takes the alphanumeric parameter of the data structure
07950L* describing the ticket request, the requestor's name, credit
08000L* card number, and so on. The library code file title is
08050L* OBJECT/EXAMPLE/PASCAL/LIBRARY.
08100L*
08150LIBRARY NAME CREDITLIB T OBJECT/EXAMPLE/PASCAL/LIBRARY
08200LIBRARY SUBR CREDITCHK I
08250LIBRARY PARAM <info> *
08300$$ PAGE
08350E*
08400E*----- Vector Declarations -----
08450E*
08500E* Declare a compile-time vector POSITN and two run-time vectors,
08550E* ROW and COL. POSITN is used to initialize both ROW and COL.
08600E* ROW and COL are then used for cursor positioning on output.
08650E*
08700E POSITN 1 2 40
08750E ROW 80 1
08800E COL 80 1
08850E*
08900E* Declare a compile-time vector with message text. The text is
08950E* in the alternate vector; the primary vector is just used to
09000E* mark the relative element number of each message at
09050E* compile time and is not actually used at run time.
09100E*
09150E MSGNUM 1 10 4 MSG 60

```

```

.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
09200$$ PAGE
09250I*
09300I*----- Input Specifications -----
09350I*
09400IDUMMY    NS  01
09450I                                1  20 DUMREC
09500I*
09550IRDS      06                                *ALL
09600I*
09650IEVENTS   07                                *ALL
09700I*
09750I* I-spec for COMS input header INHDR.
09800I*
09850I* Record type for ticket sale screen (put first because it is
09900I* most common).
09950I*
10000IINHDR    AA  15   2 CS   3 CE
10050I                                1   1 QUIT
10100I                                2   3 KEY
10150I                                4 150 REQUEST
10160I                                1 150 REC
10200I*
10250I*      Record type for HOME screen.
10300I*
10350I      AA  12   2 CH   3 CO
10400I                                1   1 QUIT
10450I                                2   3 KEY
10500I                                4   4 CHOICE
10520I                                1 150 REC
10550I*
10600I*      Record type for ADD screen.
10650I*
10700I      AA  13   2 CA   3 CD
10750I                                1   1 QUIT
10800I                                2   3 KEY
10850I                                4  23 E-NAME
10900I                                24 290E-DATE
10950I                                30 342E-PRCE
11000I                                35 400E-CAP
11020I                                1 150 REC

```

```

.....*.....1....*.....2....*.....3....*.....4....*.....5....*.....6....*.....7..
11050I*
11100I*      Record type for DELETE screen.
11150I*
11200I      AA  14   2 CD   3 CE
11250I                                     1   1 QUIT
11300I                                     2   3 KEY
11350I                                     4  23 E-NAME
11400I                                     24 290E-DATE
11420I                                     1 150 REC
11450I*
11500I* Record type for unexpected screens, such as when a terminal
11550I* has been turned off and on, erasing the key field from memory.
11600I*
11650I      AA  11
11700I                                     1   1 QUIT
11750I                                     2   3 KEY

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7...
11800$$ PAGE
11850I*
11900I*----- Data Structures -----
11950I*
12000I*-----
12050I* Definition of the data structure REQUEST. REQUEST represents a
12100I* ticket request. It contains the following information:
12150I*
12200I*   Field Name      Meaning
12250I*   -----
12300I*   CUSNAM          Customer's name. LNAME, FNAME, MNAME are
12350I*                   subfields of CUSNAM.
12400I*   CUSADR          Customer's address. STADR, APT, CITY, STATE,
12450I*                   ZIP are subfields of CUSADR.
12500I*   CARD            Credit card being used (VISA, and so on).
12550I*   CARDNO           Credit card number.
12600I*   CARDEX          Credit card expiration date (YYMM).
12650I*   EVENT           Name of requested event.
12700I*   DATE            Date (in form YYMMDD) of requested event.
12750I*   QUANT           Number of requested seats.
12800I*-----
12850I REQUEST      DS
12900I              1  41 CUSNAM
12950I              1  20 LNAME
13000I             21  40 FNAME
13050I             41  41 MNAME
13100I             42  97 CUSADR
13150I             42  67 STADR
13200I             68  70 APT
13250I             71  90 CITY
13300I             91  92 STATE
13350I             93  97 ZIP
13400I             98 101 CARD
13450I            102 114 CARDNO
13500I            115 118 CARDEX
13550I            119 138 EVENT
13600I            139 144 DATE
13650I            145 147 QUANT

```

```

.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7.
13700$$ PAGE
13750C*
13800C*----- Calculation Specifications -----
13850C*
13900C                EXSR INIT
13950C*
14000C* The main processing loop. The program does not terminate unless
14050C* instructed to by COMS. If terminated, the tag EXIT is used to
14100C* exit the Calculation Specifications.
14150C*
14200C                RECV        TAG
14250C*
14300C* Erase the response sent for the previous message.
14350C*
14400C                MOVE MSG,5      RESP    60      msg,5=blank
14450C                RECV INHDR              LR
14500C    LR          GOTO EXIT
14550C                EXSR PROCES
14600C                GOTO RECV
14650C*
14700C* Tag used as destination when terminating program.
14750C*
14800C                EXIT        TAG
14850C*
14900C*----- Subroutines -----

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
14950$$ PAGE
15000C*
15050C*=====
15100C* Subroutine to initialize program data.
15150C*=====
15200C*
15250CSR          INIT          BEGSR
15300C*          ****
15350C*
15400C* Enable the program ONLINE.
15450C*
15550C                      EXSR ONLINE
15600C*
15650C* Link to CREDITLIB.
15700C*
15750C          LINK          TAG
15800C                      EEEXSR$LINKLIB          19
15850C                      PARAMCREDITLIB
15900C  19          MSG,10      DSPLYODT          OK          2
15950C  19                      GOTO LINK
16000C*
16050C* Initialize screen control characters.
16100C*
16150C                      MOVEAPOSITN      ROW
16200C                      MOVEAPOSITN      COL
16250C                      MOVE @0C@      HCLEAR  1      Home / clear
16300C                      MOVE @11@      DC1    1      Write transpa
16350C                      MOVE @1E@      ENDHI   1      End highlight
16400C                      MOVE @27E6@     FORMON  2      Enble forms m
16450C                      MOVE @1F@      STFLD   1      Start form fi
16500C                      MOVE @18@      BLINK   1      Blinking char
16550C                      MOVE @3F@      BRIGHT  1      Bright chars
16600C                      MOVE @0D@      CR      1      Carriage retu
16650C                      MOVE @3C@      HOME    1      Home cursor
16700C                      MOVE @1C@      PRTR    1      Protect/trans
16750C                      MOVE @0E@      REVVID  1      Reverse video
16800C                      MOVE @19@      SECURD  1      Secured video
16850C                      MOVE @0F@      ULINE   1      Highlight, _
16900C                      MOVE @03@      ETX     1      End of text
16950C                      MOVE @274D@     XMT     2      Transmit
17000C                      MOVE @277F@     MOVE    2      Move cursor
17050CSR                      ENDSR                      * END INIT

```

```

.....1....*....2....*....3....*....4....*....5....*....6....*....7..
17100$$ PAGE
17150C*
17200C*=====
17250C* Subroutine to process a received screen (central routine).
17300C*=====
17350C*
17400CSR      PROCES      BEGSR
17450C*      *****
17500C*
17550C* One of the record-identifying indicators (columns 11-15) has
17600C* been set to ON by the RECV. Each indicates a different request.
17650C*
17700C  15          EXSR TICKET
17750C  14          EXSR DELETE
17800C  13          EXSR ADD
17850C  12          EXSR PICK
17900C  11          EXSR HOME
17950C          SETOF          111213
18000C          SETOF          1415
18050CSR          ENDSR          * END PROCES

```

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7...
18100$$ PAGE
18150C*
18200C*=====
18250C* Subroutine to process a ticket request.
18300C*
18350C* Assumptions:
18400C*
18450C* - The data structure REQUEST contains a record received from
18500C* TICKETDB window, either from online transmission or from
18550C* recovery. The request is for ticket purchase.
18600C*
18650C* Processing Steps:
18700C*
18750C* 1. Check the customer credit (indicator 50).
18800C* If credit is bad, store "bad credit" as response, and
18850C* skip steps 2 and 3.
18900C* 2. LOCK the record in the database representing the
18950C* requested event and date.
19000C* 3. If the event record is found, check whether there are
19050C* enough seats left to fill this request. If so, increase
19100C* the sold count, print the confirmation notice, and note
19150C* "tickets will be delivered" in response field.
19200C* If insufficient tickets remain to process the request,
19250C* store "sold out" as response.
19300C* 4. SEND new SELL screen which contains the response
19350C* to this request.
19400C*=====
19450C*
19500CSR          TICKET      BEGSR
19550C*          *****
19600C*
19650C* Check the QUIT field. If a nonblank value is present, return
19700C* the operator to the HOME screen.
19750C*
19800C          QUIT          COMP ' '          20
19850C N20          EXSR HOME
19900C N20          GOTO TIKXIT
19950C*
20000C* Enter transaction state, auditing the record most recently
20050C* received from INHDR ("REC"), plus the header INHDR itself. Note
20100C* that the restart designator must be moved into the appropriate
20150C* field of the restart record that is audited at TREND.
20160C* NOTE: If level 3.7 DMSII is used, then the Factor 2 field can
20170C* be left blank and the explicit move of RESTD to RDSLOC can
20180C* be omitted. These actions are performed automatically by
20190C* 3.7 DMSII.
20200C*
20250C          INHDR          TRBEGREC
20300C          MOVE RESTD          RDSLOC

```

....*....1....*....2....*....3....*....4....*....5....*....6....*....7..

20350C*

20400C* Check credit.

20450C*

20500C

EEXSRCREDITCHK IN50

20550C

PARAM

REQUEST

20600C N50

MOVEMSG,1

RESP

* bad credit

20650C N50

GOTO ENDTIK

20700C*

20750C* Find the event for which the ticket is requested.

20800C*

20850C

EDATES

LOCK EVENTS

RD1

20900C

DMKEYDATE

E-DATE

EQL

20950CAN

DMKEYEVENT

E-NAME

EQL

21000C*

21050C* If the event is not in the database, report.

21100C*

21150C DA

MOVEMSG,2

RESP

* not there

21200C DA

GOTO ENDTIK

21250C*

21300C* If an error other than NOTFOUND occurred, terminate.

21350C*

21400C D1

DMTER

* DMTERMINATE

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
21450C*
21500C* Are there enough seats to fill the request?
21550C*
21600C          E-CAP      SUB  E-SOLD      T-LEFT  60
21650C          T-LEFT      COMP QUANT                21
21700C*
21750C* Not enough seats - report.
21800C*
21850C  21          MOVEMSG,3      RESP          * sold out
21900C  21          GOTO ENDTIK
21950C*
22000C* Enough seats - fill request and add the number of seats sold to
22050C* the sold count in the database.
22100C*
22150C          E-PRCE      MULT QUANT      DUE      72
22200C          Z-ADDQUANT      TCOUNT  50
22250C          SETON                94
22300C          EXCPTPRINT
22350C          SETOF                94
22400C          ADD QUANT      E-SOLD
22450C          SETON                71
22500C          STOREEVENTS
22550C          SETOF                71
22600C          MOVEMSG,4      RESP          * complete
22650C*
22700C          ENDTIK      TAG
22750C*
22800C* Exit transaction state, simultaneously sending a new SELL screen
22850C* containing the response to this request.
22900C*
22950C          SETON                82
23000C          OUTHDR      TRENDSEND
23050C          SETOF                82
23100CSR      TIKXIT      ENDSR          * END TICKET

```

```

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..
23150$$ PAGE
23200C*
23250C*=====
23300C* Subroutine to show what events are in the database. This
23350C* subroutine is invoked when the user places a "?" in the CHOICE
23400C* field of the HOME screen.
23450C*
23500C* Processing Steps:
23550C*
23600C*     1. Clear screen.
23650C*     2. Find the first event in the database and SEND if found.
23700C*     3. Find next event and SEND until a NOTFOUND occurs. (SEND
23750C*        uses carriage return, not home-clear.)
23800C*     4. SEND notice to screen that the operator should transmit
23850C*        a blank to get a fresh HOME screen.
23900C*=====
23950C*
24000CSR          SHOW          BEGSR
24050C*          ****
24100C*
24150C* Check the QUIT field. If a nonblank value is present, return
24200C* the operator to the HOME screen.
24250C*
24300C          QUIT          COMP ' '          20
24350C N20          EXSR HOME
24400C N20          GOTO SHOXIT
24450C*
24500C* Clear screen.
24550C*
24600C          SETON          90
24650C          SEND OUTHDR
24700C          SETOF          90
24750C*
24800C* Start count of events found at zero.
24850C*
24900C          Z-ADD0          ECOUNT 100
24950C*
25000C* Find the first event.
25050C*
25100C          EDATES          FIND EVENTS          FIRST          SD1

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7...
25150C*
25200C* Tag for top of show loop.
25250C*
25300C          SHOMOR    TAG
25350C*
25400C* If no events are in the database, quit.
25450C*
25500C    DA              GOTO ENDSHO
25550C*
25600C* If an error other than NOTFOUND occurred, terminate.
25650C*
25700C    D1              DMTER                * DMTERMINATE
25750C*
25800C* Event found. Count and report.
25850C*
25900C          ADD 1          ECOUNT
25950C          SETON                      91
26000C          SEND OUTHDR
26050C          SETOF                      91
26100C*
26150C* Find another event.
26200C*
26250C          EDATES    FIND EVENTS    NEXT    SD1
26300C          GOTO SHOMOR
26350C*
26400C* Tag for showing list completed.
26450C*
26500C          ENDSHO    TAG
26550C          SETON                      92
26600C          SEND OUTHDR
26650C          SETOF                      92
26700CSR    SHOXIT    ENDSR                * END SHOW

```

```

.....*.....1....*.....2....*.....3....*.....4....*.....5....*.....6....*.....7..
26750$$ PAGE
26800C*
26850C*
26900C*=====
26950C* Subroutine to delete an event from the database.
27000C*
27050C* Processing Steps:
27100C*
27150C*      1. LOCK the record in the database representing the event
27200C*          to be deleted.
27250C*      2. If the event record is found, delete it and store the
27300C*          response "event deleted". If it is not found, store
27350C*          response "not deleted".
27400C*      3. SEND new DELETE screen, which contains the response to
27450C*          this request.
27500C*=====
27550C*
27600CSR      DELETE      BEGSR
27650C*          *****
27700C*
27750C* Check the QUIT field. If a nonblank value is present, return
27800C* the operator to the HOME screen.
27850C*
27900C      QUIT      COMP ' '      20
27950C N20      EXSR HOME
28000C N20      GOTO DELXIT
28050C*
28060C* Enter transaction state and audit the record most recently
28080C* received from INHDR ("REC") plus the header INHDR itself. Note
28100C* that the restart designator must be moved into the appropriate
28120C* field of the restart record that is audited at TREND.
28140C* NOTE: If level 3.7 DMSII is used, then the Factor 2 field can
28160C* be left blank and the explicit move of RESTD to RDSLOC can
28180C* be omitted. These actions are performed automatically by
28200C* 3.7 DMSII.
28220C*
28240C      INHDR      TRBEGREC
28400C      MOVE RESTD      RDSLOC

```

```

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..
28450C*
28500C* Find the event to be deleted.
28550C*
28600C          EDATES      LOCK EVENTS          RD1
28650C          DMKEYE-DATE  E-DATE      EQL
28700CAN          DMKEYE-NAME  E-NAME      EQL
28750C*
28800C* If the event is not in the database, report.
28850C*
28900C  DA          MOVEMSG,6      RESP          * not found
28950C  DA          GOTO ENNDEL
29000C*
29050C* If an error other than NOTFOUND occurred, terminate.
29100C*
29150C  D1          DMTER          * DMTERMINATE
29200C*
29250C* Event found. Delete it and report.
29300C*
29350C          DELETEDEVENTS          S
29400C          MOVEMSG,7      RESP          *delete event
29450C          ENNDEL      TAG
29500C          SETON          84
29550C          OUTHDR      TRENDSEND
29600C          SETOF          84
29650CSR      DELXIT      ENDSR          * END DELETE

```

```

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..
29700$$ PAGE
29750C*
29800C*=====
29850C* Subroutine to add an event to the database.
29900C*
29950C* Assumptions:
30000C*
30050C* - The data structure REQUEST contains a record received from
30100C*   the TICKETDB window, either from online transmission or from
30150C*   recovery. The request is to add an event to the database.
30200C*
30250C* Processing Steps:
30300C*
30350C*   1. Set E-SOLD (# of tickets sold) to zero. E-SOLD is a local
30400C*       field as well as a field of the data set EVENTS, and can
30450C*       have a value lingering from a previous transaction. The
30500C*       other fields of EVENTS are set when an ADD record is input.
30550C*   2. STORE the record into the database.
30600C*   3. Check for duplicates error and store as the response to
30650C*       whether the event was successfully added or not.
30700C*   4. SEND new ADD screen containing the response to this
30750C*       request.
30800C*=====
30850C*
30900CSR          ADD          BEGSR
30950C*          ***
31000C*
31050C* Check the QUIT field. If a nonblank value is present, return
31100C* the operator to the HOME screen.
31150C*
31200C          QUIT          COMP ' '          20
31250C N20          EXSR HOME
31300C N20          GOTO ADDXIT

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
31350C*
31360C* Enter transaction state, auditing the record most recently
31380C* received from INHDR ("REC") plus the header INHDR itself.
31400C* The restart designator must be moved into the appropriate
31420C* field of the restart record that is audited at TREND.
31440C* NOTE: If level 3.7 DMSII is used, then the Factor 2 field can
31460C* be left blank and the explicit move of RESTD to RDSLOC can
31480C* be omitted. These actions are performed automatically by
31500C* 3.7 DMSII.
31520C*
31540C          INHDR      TRBEGREC
31700C          MOVE RESTD      RDSLOC
31750C          Z-ADD0          E-SOLD
31800C          SETON                      70
31850C          STOREEVENTS          01
31900C          SETOF                      70
31950C*
32000C* If the event was already in the database, store as response.
32050C*
32100C  DB          MOVEMSG,8      RESP          * duplicate
32150C  DB          GOTO ENDADD
32200C*
32250C* If an error other than NOTFOUND occurred, terminate.
32300C*
32350C  D1          DMTER                      * DMTERMINATE
32400C*
32450C* Event added. Store as response.
32500C*
32550C          MOVEMSG,9      RESP          * event added
32600C*
32650C          ENDADD      TAG
32700C*
32750C* Exit transaction state, sending a fresh ADD screen that also
32800C* contains the response to this one.
32850C*
32900C          SETON                      83
32950C          OUTHDR      TRENDSEND
33000C          SETOF                      83
33050CSR      ADDXIT      ENDSR          * END ADD

```

```

.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
33100$$ PAGE
33150C*
33200C*=====
33250C* Subroutine to send the action screen requested from the HOME
33300C* screen. This input record does not need to be audited; no
33350C* database change takes place. The record is not needed to replay
33400C* other transactions during recovery.
33450C*=====
33500C*
33550CSR          PICK      BEGSR
33600C*          ****
33650C*
33700C* Check the QUIT field. If a nonblank value is present, close
33750C* the window for the user. Then resume waiting on RECV operation.
33800C*
33850C          QUIT      COMP ' '          20
33900C N20          SETON          80
33950C N20          SEND OUTHDR
34000C N20          SETOF          80
34050C N20          GOTO RECV
34100C*
34150C* Compare the CHOICE field from the screen received to the known
34200C* values and send the appropriate screen. If an unrecognized key
34250C* was entered, send the HOME screen so the operator can try again.
34300C*
34350C          CHOICE    COMP 'S'          20* SELL
34400C 20          SETON          82
34450C 20          SEND OUTHDR
34500C 20          SETOF          82
34550C 20          GOTO ENDPIK
34600C*
34650C          CHOICE    COMP 'A'          20* ADD
34700C 20          SETON          83
34750C 20          SEND OUTHDR
34800C 20          SETOF          83
34850C 20          GOTO ENDPIK
34900C*
34950C          CHOICE    COMP 'D'          20* DELETE
35000C 20          SETON          84
35050C 20          SEND OUTHDR
35100C 20          SETOF          84
35150C 20          GOTO ENDPIK
35200C*

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
35250C*
35300C          CHOICE    COMP '?'          20* INQUIRY
35350C    20          EXSR SHOW
35400C    20          GOTO ENDPIK
35450C*
35500C          SETON          85* OTHER
35550C          SEND OUTHDR
35600C          SETOF          85
35650C*
35700CSR    ENDPIK    ENDSR          * END PICK
35750$$$ PAGE
35800C*
35850C*=====
35900C* Subroutine to send the HOME screen. This is done whenever
35950C* unrecognized input is received.
36000C*=====
36050C*
36100CSR    HOME      BEGSR
36150C*    ****
36200C*
36250C          SETON          85* Send HOME
36300C          SEND OUTHDR
36350C          SETOF          85
36400C*
36450CSR          ENDSR          * END HOME
36500$$$ PAGE
36550C*
36600C*=====
36650C* Subroutine to put the program in online mode.
36700C*=====
36750C*
36800CSR    ONLINE    BEGSR
36850C*    *****
36900C*
36950C    'ONLINE'  ENBLEINHDR
37000C          MOVE PROGD    RDSPRG
37050C          MOVE RESTD    RDSLOC
37100CSR          ENDSR          * END ONLINE

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
37150$$ PAGE
37200C*
372500*
373000*-----Output-Format Specifications -----
373500*
374000*-----
374500* Output-Format Specifications for restart data set.
375000*-----
375500RDS      R
376000          RDS-ID          'ONLINE'
376500          RDSPRG          RDSPRG
377000          RDSLOC          RDSLOC
377500*-----
378000* Output-Format Specifications for data set EVENTS.
378500*-----
379000EVENTS  SADD      70          *ALL
379500EVENTS  S        71          *ALL

```

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
38000$$ PAGE
380500*-----
381000* Output-Format Specifications for output header OUTHDR.
381500*-----
382000*
382500*          *****
383000*          **  SELL SCREEN  **
383500*          *****
384000*
384500* Indicator 82 is set to ON when a SELL form is to be transmitted.
385000OUTHDR  E      82
385500          HCLEAR      1
386000          STFLD       2
386500          3 ' '
387000          ENDHI       4
387500          13 ' <-- quit'
388000          MOVE        15
388500          COL,75      16
389000          ROW,1       17
389500          SECURD      18
390000          REVVID      19
390500          PRTR        20
391000          23 'SE '
391500          ENDHI       23
392000*
392500          MOVE        26
393000          COL,32      27
393500          ROW,3       28
394000          BRIGHT      29
394500          49 'SELL TICKETS '
395000*
395500* Add bright partition.
396000*
396500          MOVE        51
397000          COL,1       52
397500          ROW,4       53
398000          ULINE       54
398500*
399000          MOVE        56
399500          COL,4       57
400000          ROW,6       58
400500          78 'Please enter custome'
401000          98 'r and event data: '

```

RPG Application Program Using Language Extensions (cont.)

....*....1....*....2....*....3....*....4....*....5....*....6....*....7..

```

401500*
402000          MOVE      100
402500          COL,10    101
403000          ROW,8     102
403500          122 'Customer name:  '
404000          STFLD     123
404500          143 'last          '
405000          ENDHI     144
405500          STFLD     145
406000          165 'first          '
406500          ENDHI     166
407000          STFLD     167
407500          168 ' '
408000          ENDHI     169
408500*
409000          MOVE      171
409500          COL,10    172
410000          ROW,10    173
410500          193 'Customer address:  '
411000          STFLD     194
411500          200 '#'          '
412000          ENDHI     201
412500          202 ' '
413000          STFLD     203
413500          223 'street          '
414000          ENDHI     224
414500          229 ' apt '
415000          STFLD     230
415500          233 ' '
416000          ENDHI     234
416500*
417000          MOVE      236
417500          COL,30    237
418000          ROW,11    238
418500          STFLD     239
419000          259 'city          '
419500          ENDHI     260
420000          261 ' '
420500          STFLD     262
421000          264 ' '
421500          ENDHI     265
422000          266 ' '
422500          STFLD     267
423000          272 ' '
423500          ENDHI     273

```

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
424000*
424500          MOVE      275
425000          COL,10    276
425500          ROW,13    277
426000          297 'Credit card:      '
426500          STFLD     298
427000          302 '      '
427500          ENDHI     303
428000          323 '      (VISA, MC, AMEX)'
428500          MOVE      325
429000          COL,10    326
429500          ROW,14    327
430000          347 '      #:      '
430500          STFLD     348
431000          352 '      '
431500          ENDHI     353
432000          STFLD     354
432500          357 '      '
433000          ENDHI     358
433500          STFLD     359
434000          362 '      '
434500          ENDHI     363
435000          STFLD     364
435500          367 '      '
436000          ENDHI     368
436500*
437000          MOVE      370
437500          COL,10    371
438000          ROW,15    372
438500          392 'Expiration date:    '
439000          STFLD     393
439500          395 'YY'
440000          ENDHI     396
440500          STFLD     397
441000          399 'MM'
441500          ENDHI     400
442000*
442500          MOVE      402
443000          COL,10    403
443500          ROW,17    404
444000          424 'Event name:      '
444500          STFLD     425
445000          MOVE      427
445500          COL,51    428
446000          ROW,17    429
446500          ENDHI     430

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
447000*
447500          MOVE      432
448000          COL,10    433
448500          ROW,19    434
449000          454 'Event date: '
449500          STFLD     455
450000          457 'YY'
450500          ENDHI     458
451000          STFLD     459
451500          461 'MM'
452000          ENDHI     462
452500          STFLD     463
453000          465 'DD'
453500          ENDHI     466
454000*
454500          MOVE      468
455000          COL,10    469
455500          ROW,21    470
456000          490 'Number of tickets: '
456500          STFLD     491
457000          494 '002'
457500          ENDHI     495
458000*
458500* Display confirmation of last action.
459000*
459500          MOVE      497
460000          COL,1     498
460500          ROW,24    499
461000          BRIGHT    500
461500          RESP      560
462000*
462500* Add bottom bright partition.
463000*
463500          MOVE      562
464000          COL,1     563
464500          ROW,23    564
465000          ULINE     565
465500*
466000* Turn on forms mode and mark the end of text.
466500*
467000          FORMON     567
467500          568 ' '
468000          ETX       569

```

RPG Application Program Using Language Extensions (cont.)

```

.....1....*....2....*....3....*....4....*....5....*....6....*....7..
46850$$ PAGE
469000*
469500*
470000*
470500*
471000*
471500* Set indicator 83 to on when ADD event form is to be transmitted.
472000OUTHDR E      83
472500          HCLEAR      1
473000          STFLD       2
473500          3 ' '
474000          ENDHI       4
474500          13 ' <-- quit'
475000          MOVE        15
475500          COL,75      16
476000          ROW,1       17
476500          SECURD      18
477000          REVVID      19
477500          PRTR        20
478000          23 'AD '
478500          ENDHI       23
479000*
479500          MOVE        26
480000          COL,32      27
480500          ROW,4       28
481000          BRIGHT      29
481500          49 '      ADD EVENT      '
482000*
482500* Add bright partition.
483000*
483500          MOVE        51
484000          COL,1       52
484500          ROW,5       53
485000          ULINE       54
485500*
486000          MOVE        56
486500          COL,5       57
487000          ROW,8       58
487500          78 'Please enter data fo'
488000          98 'r event to be added:'

```

RPG Application Program Using Language Extensions (cont.)

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
488500*
489000          MOVE      100
489500          COL,10    101
490000          ROW,11    102
490500          122 'Event name:      '
491000          STFLD     123
491500          MOVE      125
492000          COL,51    126
492500          ROW,11    127
493000          ENDHI     128
493500*
494000          MOVE      130
494500          COL,10    131
495000          ROW,13    132
495500          152 'Event date:      '
496000          STFLD     153
496500          155 'YY'
497000          ENDHI     156
497500          STFLD     157
498000          159 'MM'
498500          ENDHI     160
499000          STFLD     161
499500          163 'DD'
500000          ENDHI     164
500500*
501000          MOVE      166
501500          COL,10    167
502000          ROW,15    168
502500          188 'Event price:     '
503000          STFLD     189
503500          192 'ddd'
504000          ENDHI     193
504500          STFLD     194
505000          196 'cc'
505500          ENDHI     197
506000*
506500          MOVE      199
507000          COL,10    200
507500          ROW,17    201
508000          221 'Event capacity:   '
508500          STFLD     222
509000          228 '123456'
509500          ENDHI     229

```

RPG Application Program Using Language Extensions (cont.)

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..

510000*

510500* Display confirmation of last action.

511000*

511500	MOVE	231
--------	------	-----

512000	COL,1	232
--------	-------	-----

512500	ROW,24	233
--------	--------	-----

513000	BRIGHT	234
--------	--------	-----

513500	RESP	294
--------	------	-----

514000*

514500* Add bottom bright partition.

515000*

515500	MOVE	296
--------	------	-----

516000	COL,1	297
--------	-------	-----

516500	ROW,23	298
--------	--------	-----

517000	ULINE	299
--------	-------	-----

517500*

518000* Turn on forms mode and mark the end of text.

518500*

519000	FORMON	301
--------	--------	-----

519500		302 ' '
--------	--	---------

520000	ETX	303
--------	-----	-----

.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..

52050\$\$ PAGE

521000*

521500*

522000*

** DELETE SCREEN **

522500*

523000*

523500* Set indicator 84 to ON when a DELETE form is to be transmitted.

524000OUTHDR E 84

524500

HCLEAR 1

525000

STFLD 2

525500

3 ' '

526000

ENDHI 4

526500

13 ' <-- quit'

527000

MOVE 15

527500

COL,75 16

528000

ROW,1 17

528500

SECURD 18

529000

REVVID 19

529500

PRTR 20

530000

23 'DE '

530500

ENDHI 23

531000*

531500

MOVE 26

532000

COL,32 27

532500

ROW,4 28

533000

BRIGHT 29

533500

49 'DELETE EVENT'

534000*

534500* Add bright partition.

535000*

535500

MOVE 51

536000

COL,1 52

536500

ROW,5 53

537000

ULINE 54

537500*

538000

MOVE 56

538500

COL,5 57

539000

ROW,8 58

539500

78 'Please identify even'

540000

98 't to be deleted: '

RPG Application Program Using Language Extensions (cont.)

....*....1....*....2....*....3....*....4....*....5....*....6....*....7..

```

540500*
541000          MOVE      100
541500          COL,10    101
542000          ROW,11    102
542500          122 'Event name:      '
543000          STFLD     123
543500          MOVE      125
544000          COL,51    126
544500          ROW,11    127
545000          ENDHI     128
545500*
546000          MOVE      130
546500          COL,10    131
547000          ROW,13    132
547500          152 'Event date:      '
548000          STFLD     153
548500          155 'YY'
549000          ENDHI     156
549500          STFLD     157
550000          159 'MM'
550500          ENDHI     160
551000          STFLD     161
551500          163 'DD'
552000          ENDHI     164
552500*
553000* Display confirmation of last action.
553500*
554000          MOVE      166
554500          COL,1     167
555000          ROW,24    168
555500          BRIGHT    169
556000          RESP      229
556500*
557000* Add bottom bright partition.
557500*
558000          MOVE      231
558500          COL,1     232
559000          ROW,23    233
559500          ULINE     234
560000*
560500* Turn on forms mode and mark the end of text.
561000*
561500          FORMON     236
562000          237 ' '
562500          ETX       238

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
56300$$ PAGE
563500*
564000* *****
564500* ** HOME SCREEN **
565000* *****
565500*
566000* Indicator 85 is set to ON when a HOME form is to be transmitted.
5665000UTHDR E 85
567000 HCLEAR 1
567500 STFLD 2
568000 3 ' '
568500 ENDHI 4
569000 13 ' <-- quit'
569500 MOVE 15
570000 COL,75 16
570500 ROW,1 17
571000 SECURD 18
571500 REVVID 19
572000 PRTR 20
572500 23 'HO '
573000 ENDHI 23
573500*
574000 MOVE 26
574500 COL,32 27
575000 ROW,4 28
575500 BRIGHT 29
576000 49 'TICKET DATABASE '

```

RPG Application Program Using Language Extensions (cont.)

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..

576500*

577000* Add bright partition.

577500*

578000 MOVE 51

578500 COL,1 52

579000 ROW,5 53

579500 ULINE 54

580000*

580500 MOVE 56

581000 COL,5 57

581500 ROW,8 58

582000 82 'Please select action: '

582500 STFLD 83

583000 84 ' '

583500 ENDHI 85

584000*

584500 MOVE 87

585000 COL,15 88

585500 ROW,11 89

586000 109 'A = Add event '

586500 MOVE 111

587000 COL,15 112

587500 ROW,13 113

588000 133 'D = Delete event '

588500 MOVE 135

589000 COL,15 136

589500 ROW,15 137

590000 157 'S = Sell tickets '

590500*

591000* Add bottom bright partition.

591500*

592000 MOVE 159

592500 COL,1 160

593000 ROW,23 161

593500 ULINE 162

594000*

594500* Turn on forms mode and mark the end of text.

595000*

595500 FORMON 164

596000 165 ' '

596500 ETX 166

RPG Application Program Using Language Extensions (cont.)

```

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..
59700$$ PAGE
597500*
598000*
598500*
599000*
599500*
600000* Indicator 80 is set to ON when the operator wants to sign off
600500* the window. Give a "goodbye" and put '?CLOSE' at the top of the
601000* screen. Note that the XMT after the ?CLOSE is commented out.
601500* If the COMS used to run the program is a secondary MCS,
602000* two question marks are needed. Uncommenting the record causes
602500* the ?CLOSE to actually be transmitted back.
603000*
6035000UTHDR E      80
604000
604500
605000
605500
606000
606500
607000
607500*
608000

```

	HCLEAR	1
	MOVE	3
	COL,33	4
	ROW,12	5
		15 'GOOD BYE. '
	HOME	16
		26 '?CLOSE '
	XMT	28
	ETX	29

RPG Application Program Using Language Extensions (cont.)

```

.....*....1....*....2....*....3....*....4....*....5....*....6....*....7..
60850$$ PAGE
609000*
609500* *****
610000* ** SHOW SCREENS **
610500* *****
611000*
611500* Indicator 90 is set to ON to clear the screen.
612000OUTHDR E 90
612500 HCLEAR 1
613000 ETX 2
613500*
614000* Indicator 91 is set to ON to show an event from the database.
614500OUTHDR E 91
615000 DC1 1
615500 13 'Event name ='
616000 21 '>'
616500 E-NAME 41
617000 42 '<'
617500 67 'Price ='
618000 E-PRCEJ 81
618500 93 'Event date ='
619000 E-DATEY 107
619500 125 'Capacity ='
620000 E-CAP 1 137
620500 147 'Sold ='
621000 E-SOLD1 161
621500 181 '-----'
622000 CR 182
622500 ETX 183
623000*
623500* Indicator 92 is set to ON when done showing events.
624000OUTHDR E 92
624500 DC1 1
625000 16 '*****'
625500 ECOUNTZ 25
626000 40 'Events found.'
626500 61 'XMT blank for home.'
627000 77 '*****'
627500 CR 78
628000 ETX 79

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
62850$$ PAGE
629000*-----
629500* Output-Format Specifications for file PRINT.
630000*-----
630500*
631000PRINT    E          94
631500
632000
632500
633000
633500          E          94
634000          TCOUNTZ    5
634500          19 'Tickets for: '
635000          EVENT       39
635500          46 'Date:'
636000          E-DATEY     55
636500          69 'Total price:'
637000          DUE    J    80
637500          E          94
638000          12 'Sold to:'
638500          FNAME       34
639000          MNAME       36
639500          LNAME       60
640000          E          94
640500          STADR       40
641000          APT         46
641500          E          94
642000          CITY       34
642500          STATE      40
643000          ZIP        50
643500          E 1        94
644000          24 '*****'
644500          48 '*****'
645000          72 '*****'
645500          96 '*****'

```

```

.....*.....1.....*.....2.....*.....3.....*.....4.....*.....5.....*.....6.....*.....7..
64600$$ PAGE
64650V*
64700V*----- Vector Specifications -----
64750V*
64800V* Vector with cursor position control characters in it.
64850V*
64900VVECTOR SEQ                                * POSITN Vector:
64950 !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGH
65000HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnop
65050V*
65100V* Vector with message text in it. The vector has data in columns
65150V* 10 through 74. The following comment line can be used to
65200V* determine the position of a character in a line of vector text
65250V* relative to the start of the vector element.
65300V*
65350V*
65400V*
65450VVECTOR SEQ                                * MSG Vector:
65500 1: Bad credit. Ticket request denied.
65550 2: Event not found in database. Ticket request denied.
65600 3: Not enough seats left to fill request. No sale.
65650 4: Sale completed. Tickets will be delivered.
65700 5:
65750 6: Event not found in database. Not deleted.
65800 7: Event found and deleted from database.
65850 8: Event already in database. Not added.
65900 9: Event successfully added to database.
6595010: OBJECT/EXAMPLE/PASCAL/LIBRARY not present. Fix & ?AX OK:

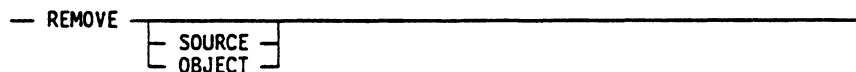
```

Appendix D

Understanding Railroad Diagrams

What Are Railroad Diagrams?

Railroad diagrams are diagrams that show you the rules for putting words and symbols together into commands and statements that the computer can understand. These diagrams consist of a series of paths that show the allowable structure, constants, and variables for a command or a statement. Paths show the order in which the command or statement is constructed. Paths are represented by horizontal and vertical lines. Many railroad diagrams have a number of different paths you can take to get to the end of the diagram. For example:



If you follow this railroad diagram from left to right, you will discover three acceptable commands. These commands are

- REMOVE
- REMOVE SOURCE
- REMOVE OBJECT

If all railroad diagrams were this simple, this explanation could end here. However, because the allowed ways of communicating with the computer can be complex, railroad diagrams sometimes must also be complex.

Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements. Railroad diagrams are intended to

- Show the mandatory items.
- Show the user-selected items.
- Present the order in which the items must appear.
- Show the number of times an item can be repeated.
- Show the necessary punctuation.

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and provides examples.

Some of the actual railroad diagrams you will encounter might be more complex. However, all railroad diagrams, simple or complex, follow the same basic rules. They all consist of paths that represent the allowable structure, constants, and variables for commands and statements.

By following railroad diagrams, you can easily understand the correct syntax for commands and statements. Once you become proficient in the use of railroad notation, the diagrams serve as quick references to the commands and statements.

Constants and Variables

A constant is an item that cannot be altered. You must enter the constant as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially underlined, you can abbreviate the constant by entering only the underlined letters. In addition to the underlined letters, any of the remaining letters can be entered. If no part of the constant is underlined, the constant cannot be abbreviated. Constants can be recognized by the fact that they are never enclosed in angle brackets (< >) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement. Variables appear in railroad diagrams enclosed in angle brackets.

In the following example, BEGIN and END are constants while <statement list> is a variable. The constant BEGIN can be abbreviated since it is partially underlined. Valid abbreviations for BEGIN are BE, BEG, and BEGI.

— BEGIN —<statement list>— END —————|

Constraints

Constraints are used in a railroad diagram to control progression through the diagram. Constraints consist of symbols and unique railroad diagram line paths. They include

- Vertical bars
- Percent signs
- Right arrows
- Required items
- User-selected items
- Loops
- Bridges

A description of each item follows.

Vertical Bar

The vertical bar symbol (|) represents the end of a railroad diagram and indicates the command or statement can be followed by another command or statement.

— SECONDWORD — (—<arithmetic expression>—) —————|

Percent Sign

The percent sign (%) represents the end of a railroad diagram and indicates the command or statement must be on a line by itself.

— STOP ————— %

Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next. A right arrow appears at the end of the first line and another right arrow appears at the beginning of the next line.

— SCALERIGHT — (—<arithmetic expression>— , —————>
-><arithmetic expression>—) —————|

Required Items

A required item can be either a constant, a variable, or punctuation. A required item appears as a single entry, by itself or with other items, on a horizontal line. Required items can also exist on horizontal lines within alternate paths or nested (lower-level) diagrams. If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

In the following example, the word EVENT is a required constant and <identifier> is a required variable:

— EVENT —<identifier>—————|

User-Selected Items

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line), none of the choices are required. A user-selected item can be either a constant, a variable, or punctuation. In the following railroad diagram, either the plus sign (+) or minus sign (-) can be entered before the required variable <arithmetic expression>, or the symbols can be disregarded because the diagram also contains an empty path.

— [+] —<arithmetic expression>—————|
— [-] —————|

Loop

A loop represents an item or group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

Some loops include a return character. A return character is a character — often a comma (,) or semicolon (;) — required before each repetition of a loop. If there is no return character, the items must be separated by one or more blank spaces.

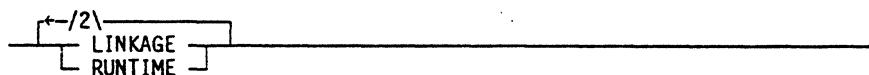
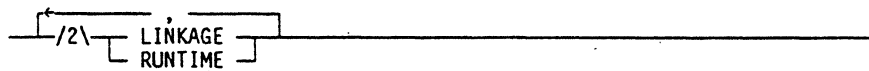


Bridge

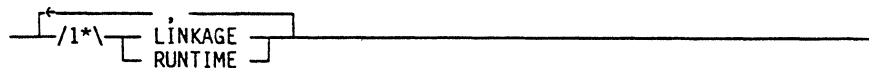
Sometimes a loop also includes a bridge, which is used to show the maximum number of times the loop can be repeated. The bridge can precede the contents of the loop, or it can precede the return character (if any) on the upper line of the loop.

The bridge determines the number of times you can cross that point in the diagram. The bridge is an integer enclosed in sloping lines (/ \). Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

In the first bridge example, you can enter LINKAGE or RUNTIME no more than two times. In the second bridge example, you can enter LINKAGE or RUNTIME no more than three times.



In some bridges an asterisk (*) follows the number. The asterisk means that you must select one item from the group.



The following figure shows the types of constraints used in railroad diagrams.

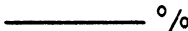
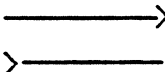
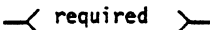
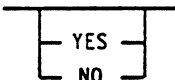
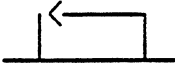
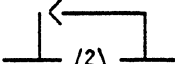
SYMBOL/PATH	EXPLANATION
	Vertical bar. Indicates that the command or statement can be followed by another command or statement.
	Percent sign. Indicates that the command or statement must be on a line by itself.
	Right arrow. Indicates that the diagram occupies more than one line.
	Required items. Indicates the constants, variables, and punctuation that must be entered in a command or statement.
	User-selected items. Indicates the items that appear one below the other in a vertical list. You select which item or items to include.
	A loop. Indicates an item or group of items that can be repeated.
	A bridge. Indicates the maximum number of times a loop can be repeated.

Figure D-1. Railroad Constraints

Following the Paths of a Railroad Diagram

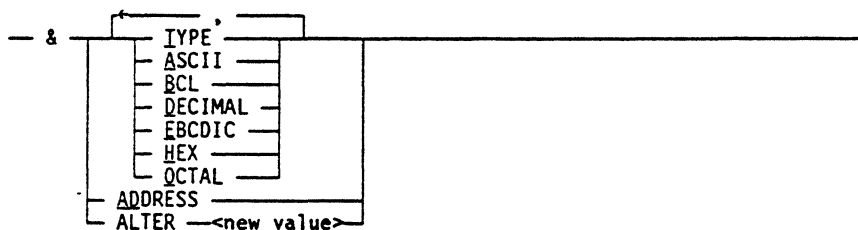
The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path, while others have several alternate paths. The following railroad diagram indicates there is only one path that requires the constant **LINKAGE** and the variable **<linkage mnemonic>**:

— LINKAGE —<linkage mnemonic>—|

Alternate paths provide choices in the construction of commands and statements. Alternate paths are provided by loops, user-selected items, or a combination of both. More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes an ampersand (&) and the constants (that are

user-selected items) in the vertical list. These constants are within a loop that can be repeated any number of times until all options have been selected. The first alternate path requires the ampersand and the required constant ADDRESS. The second alternate path requires the ampersand followed by the required constant ALTER and the required variable <new value>.



Railroad Diagram Examples with Sample Input

The following examples show five railroad diagrams and possible command and statement constructions based on the paths of these diagrams.

Example 1

<lock statement>



Sample Input

LOCK (FILE4)

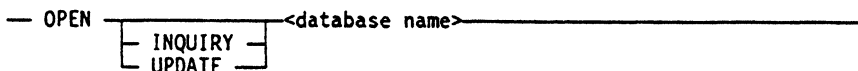
Explanation

LOCK is a constant and cannot be altered. Because no part of the word is underlined, the entire word must be entered.

The parentheses are required punctuation and FILE4 is a sample file identifier.

Example 2

<open statement>



Sample Input

OPEN DATABASE1

Explanation

The constant OPEN is followed by the variable DATABASE1, which is a database name.

The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because there is an empty path (solid line), these entries are not required.

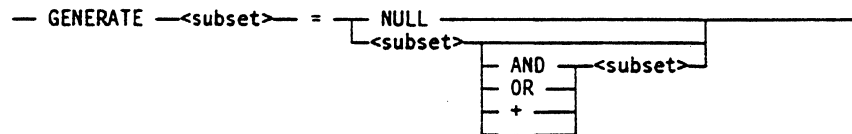
OPEN INQUIRY DATABASE1

The constant OPEN is followed by the user-selected constant INQUIRY and the variable DATABASE1.

Sample Input	Explanation
OPEN UPDATE DATABASE1	The constant OPEN is followed by the user-selected constant UPDATE and the variable DATABASE1.

Example 3

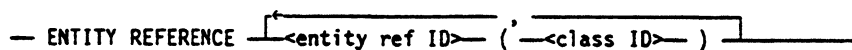
<generate statement>



Sample Input	Explanation
GENERATE Z = NULL	The GENERATE constant is followed by the variable Z, an equal sign (=), and the user-selected constant NULL.
GENERATE Z = X	The GENERATE constant is followed by the variable Z, an equal sign, and the user-selected variable X.
GENERATE Z = X AND B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the AND command (from the list of user-selected items in the nested path), and a third variable, B.
GENERATE Z = X + B	The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the plus sign (from the list of user-selected items in the nested path), and a third variable, B.

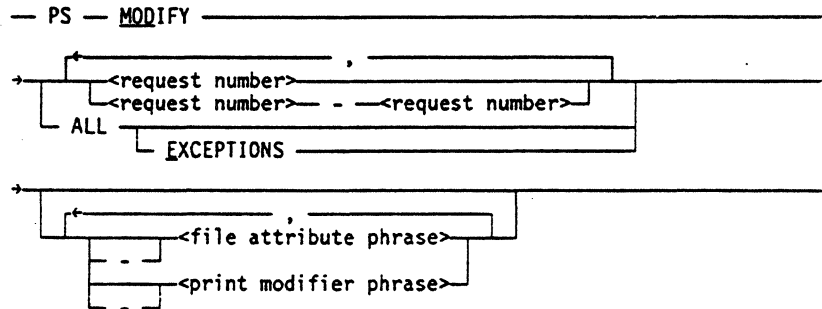
Example 4

<entity reference declaration>



Sample Input	Explanation
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)	The required item ENTITY REFERENCE is followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.
ENTITY REFERENCE ADVISOR1 (INSTRUCTOR), ADVISOR2 (ASST_INSTRUCTOR)	Because the diagram contains a loop, the pair of variables can be repeated any number of times.

Example 5



Sample Input

PS MODIFY 11159

PS MODIFY 11159,11160,11163

PS MOD 11159-11161 DESTINATION =
"LP7"

PS MOD ALL EXCEPTIONS

Explanation

The constants PS and MODIFY are followed by the variable 11159, which is a request number.

Because the diagram contains a loop, the variable 11159 can be followed by a comma, the variable 11160, another comma, and the final variable 11163.

The constants PS and MODIFY are followed by the user-selected variables 11159-11161, which are request numbers, and the user-selected variable DESTINATION = "LP7", which is a file attribute phrase. Note that the constant MODIFY has been abbreviated to its minimum allowable form.

The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.

Glossary

In this glossary, definitions taken from outside sources are preceded by an abbreviation enclosed in parentheses. Definitions from the *Vocabulary for Data Processing, Telecommunications, and Office Systems* are preceded by VDP. Unless otherwise noted, all definitions that pertain to COBOL also pertain to COBOL74 and COBOL85.

A

abort

To terminate an active program or session abnormally and, sometimes, to attempt to restart it.

ABSN

See audit block serial number.

access

To perform an action on an object. Possible actions depend on the type of object; for example, interrogating or assigning a value to a variable, reading from or writing to a file, or invoking a procedure.

Access

A logical index structure that defines the physical ordering of records in direct, ordered, and random data sets. An access functions like a set, but no physical file is associated with an access.

access mode

The manner in which records are to be operated on within a file. The two possible access modes are random and sequential.

Accessroutines

Routines that perform all physical and logical management of a database and allow many users to access the database concurrently. Each data management statement that a user language program executes invokes a portion of the Accessroutines to perform all file management functions that the statement requires.

active query

In database management, a query that the system can process. All queries activated within transaction state are deactivated at the end of transaction state.

ACTUALNAME clause

A clause that permits the renaming of a library entry point within a user program by using a different procedure identifier to link to the original library entry point name. This clause is used in ALGOL, FORTRAN77, and PL/1 application programs when a procedure is declared to be an entry point of a particular library.

ADDS

See Advanced Data Dictionary System.

Advanced Data Dictionary System (ADDS)

A software product that allows for the centralized definition, storage, and retrieval of data descriptions.

afterimage

The image of a database record in the audit trail after an update operation is performed.

ALGOL

Algorithmic language. A structured, high-level programming language that provides the basis for the stack architecture of the Unisys A Series systems. ALGOL was the first block-structured language developed in the 1960s and served as a basis for such languages as Pascal and Ada. It is still used extensively on A Series systems, primarily for systems programming.

alpha item

A data item that stores alphanumeric information (letters, numbers, special characters, and blanks) as EBCDIC characters. An alpha item cannot be used in calculations.

alphabetic character

A character that belongs to the following set of letters in either uppercase or lowercase: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space (blank).

alphanumeric character

Any character in the computer's character set.

ANSI

American National Standards Institute. A nongovernmental, nonprofit organization that is the central body for coordinating voluntary standards in the United States. ANSI also serves as the United States member of the International Standards Organization (ISO).

arithmetic expression

An expression containing any of the following: a numeric variable, a numeric elementary item, a numeric literal, identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses.

arithmetic function

A function containing calculations that produce a numeric result based on one or more records.

attribute setting entry point

In the DMINTERPRETER library, an entry point that restricts the number of times the DMINTERPRETER library can call the Accessroutines to complete a find or lock operation.

audit block serial number (ABSN)

A number associated with a block in the audit file. The first block in the first audit file is noted as ABSN 0. The ABSN is incremented by one through the last audit file.

audit trail

A file produced by the Access routines that contains various control information, including beforeimages and afterimages of records resulting from changes to the database.

audited database

A database that stores a record of changes (called the audit trail), which can be used for database recovery if a hardware or software failure occurs.

automatic subset

A subset declared with a condition that specifies which members of the data set are to be included in the subset. Entries are automatically inserted into or removed from the subset when records are added to or deleted from the data set.

B**back out**

To undo changes made against a database and to roll back the progress of one or more transactions to a previously consistent state.

beforeimage

The image of a database record in the audit trail before an update operation is performed.

binary search

A quick method of searching for records that are in a known sequence by successively halving the area to be searched.

block

(1) A group of physically adjacent records that can be transferred to or from a physical device as a group. (2) A program, or a part of a program, that is treated by the processor as a discrete unit. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

Boolean

Pertaining to variables, data items, and attributes having a value of TRUE or FALSE.

Boolean item

A data item that stores information having a logical value of TRUE or FALSE.

C**call**

A programmatic request for another procedure or program to execute.

CALL statement

In COBOL, the *CALL nonnumeric literal* statement is the only construct that enables a program to call a library entry point. The CALL statement can pass required parameters in an optional USING clause and receive a result in an optional GIVING clause.

CANDE

See Command and Edit language.

CHANGE ATTRIBUTE statement

In COBOL, the statement that is the only construct that can be used to change library attributes. When programming for the interpretive interface, this statement is used to change the **TITLE** attribute of the library to the actual name of the library object code file.

character

The actual or coded representation of a digit, letter, or special symbol in display form.

COBOL

Common Business-Oriented Language. A widely used, procedure-oriented language intended for use in solving problems in business data processing. The main characteristics of COBOL are the easy readability of programs and a considerable degree of machine independence. COBOL is the most widely used procedure-oriented language.

COBOL74

A version of the COBOL language that is compatible with the ANSI X3.23-1974 standard.

COBOL85

The latest version of the COBOL language. This version is compatible with the American National Standard, X3.23-1985.

code file

See object code file, source file.

Command and Edit language (CANDE)

A time-sharing message control system (MCS) that enables a user to create and edit files, and develop, test, and execute programs, interactively.

Communications Management System (COMS)

A general message control system (MCS) that controls online environments on A Series systems. COMS can support the processing of multiprogram transactions, single-station remote files, and multistation remote files.

compile time

The time during which a compiler analyzes program text and generates an object code file.

complex key

A key composed of multiple key items.

COMS

See Communications Management System.

condition

In COBOL, a status for which a truth value can be determined at execution time. The term *condition* (condition-1, condition-2, and so forth) implies a conditional expression consisting of either a simple condition optionally enclosed in parentheses or a combined

condition consisting of a combination of simple conditions, logical operators, and parentheses, for which a truth value can be determined. *See also* selection expression.

control file

A file containing data file coordination information, audit control information, and dynamic database parameter values.

control item

A count item, population item, or record-type item.

controlpoint

A feature that limits the amount of audit information scanned by halt/load and Abort recovery. The controlpoint is performed when the last process leaves transaction state. Halt/load and Abort recovery do not process any audit images prior to the next to last controlpoint in the audit.

count item

A control item that contains a system-maintained count of the number of counted links that refer to a record.

current path

A logical reference into a data set or set from which the current record was retrieved.

current record

The actual data set record that a program is currently referencing. Each data set has a current record, which is contained in the user work area.

current record area

See user work area.

D

DASDL

See Data and Structure Definition Language.

Data and Structure Definition Language (DASDL)

In Data Management System II (DMSII), the language used to describe a database logically and physically, and to specify criteria to ensure the integrity of data stored in the database. DASDL is the source language that is input to the DASDL compiler, which creates or updates the database description file from the input.

data independence

In data management, the property that establishes the ability to change the structural format of a database without requiring changes to or recompilation of unaffected application programs that use the database.

data item

A field in a database record or transaction format that contains a particular type of information.

data management

(1) The operating system function of placing and retrieving data in storage and protecting its security and integrity. (2) Data administration.

Data Management System (DMS)

The system responsible for storing and retrieving data while protecting data security and integrity.

Data Management System II (DMSII)

A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

data set

A collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets can contain other data sets, sets, and subsets.

data set record

A record contained in a data set, which can be accessed through a DMSII statement.

data transfer entry point

A type of entry point peculiar to the DMINTERPRETER library whereby an item needed from a data set record is moved individually to or from a variable or expression that has been declared in the application program.

data type

An interpretation applied to a string of bits. Data types can be classified as *structured* or *scalar*. Structured data types are collections of individual data items of the same or different data types, such as arrays and records. Scalar data types include real, integer, double precision, complex, logical (also called Boolean), character, pointer, and label. Most programming languages provide a declaration statement or a standard convention to indicate the data type of a variable.

database (DB)

An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single description. Ideally, all the permanent data pertinent to a particular application resides in a single database. The database is considered a global entity that several applications can access and update concurrently.

database administrator (DBA)

The person or group of people responsible for planning, designing, implementing, and maintaining a database.

database definition

A description of the logical and physical structures of a database.

database equation

The equation that refers to three operations: the specification of database titles during compilation, the run-time manipulation of database titles, and the creation of a Work

Flow Language (WFL) task equation that overrides compiled-in titles by implicitly assigning a value to the DATABASE task attribute.

database management system (DBMS)

The software used to store, retrieve, update, report on, and protect data in a database.

database name

The unique identifier of a particular database. The rules for constructing a database name are the same as those for constructing a file name.

database stack (DBS)

A stack that contains all the information necessary for the Access routines to manage a database.

DB

See database.

DBMS

See database management system.

DBS

See database stack.

deadlock

In data management, a situation in which two or more programs have locked records and are also attempting to lock records held by each other.

declaration

A programming language construct used to identify an object, such as a type or variable, to the compiler. A declaration can be used to associate a data type with the object so that the object can be used in a program.

description file

The file produced by the Data and Structure Definition Language (DASDL) or Transaction Formatting Language (TFL) compiler that contains information used when compiling all tailored software and all DMSII user-language programs for a particular database or transaction base.

direct data set

A collection of related data records stored in a file. These records are maintained in key value order. One unsigned numeric data item in the record is designated as the key item.

directory

(1) In Data Management System II (DMSII), a file with the layout for each field of the record that it describes. A directory describes the layout of records within a file. (2) In the Advanced Data Dictionary System (ADDS), a unique identifier by which one or more entities can be grouped.

directory name

A name used to refer to a group of files whose file names are identical to the directory name, except that the file names have at least one additional node following the directory name.

disjoint

Pertaining to a data set, set, or subset when it is not contained in another data set.
Contrast with embedded.

DMINTERPRETER library

The object code file within the interpretive interface that enables an application program to access a DMSII database.

DMS

See Data Management System.

DMSII

See Data Management System II.

DMSII recovery

A database routine that is initiated after a hardware, software, or operations failure while a database is in the update mode. DMSII recovery backs out any partially completed transactions by applying audit-trail images to the database to restore it to its proper state. It also passes restart information to the programs accessing the database.

DMSUPPORT library

An object code file containing the procedures that are tailored for a particular database.

double precision

Pertaining to an arithmetic value that is represented internally as a signed-magnitude mantissa and exponent and is contained in two words.

E

EBCDIC

Extended Binary Coded Decimal Interchange Code. An 8-bit code representing 256 graphic and control characters that are the native character set of most mainframe systems.

embedded

Pertaining to a data set, set, or subset contained within another data set. A record of an embedded structure must be accessed through the *master* data set in which it is embedded. *Contrast with* disjoint.

entry point

A procedure or function that is a library object.

exception

In data management, an error result returned to an application program by the data management software, explaining the reason a requested database operation was not performed.

exception handling entry point

In the DMINTERPRETER library, an entry point that aids the programmer in handling and interpreting exceptions and errors.

exception result

Synonym for result descriptor and exception word.

exception word

Synonym for result descriptor and exception result.

export

To send or carry to an outside entity the original significance or function.

expression

A combination of operands and operators that results in the generation of a single value.

F

field item

A data item that can contain unsigned integer or Boolean values.

Field Trouble Report (FTR)

A User Communication Form (UCF) used to report a problem about a product.

file equation

A mechanism for specifying the values of file attributes when a program is compiled or executed. A file equation implicitly assigns a value to the FILECARDS task attribute.

file-equate

To specify the file title and other file parameters that are different from the defaults provided by the source program.

foreign key technique

A technique for designing databases using a data item to associate records from two different data sets without using links.

FORTRAN77

A version of the FORTRAN language that is compatible with the ANSI X3.9-1978 standard.

FTR

See Field Trouble Report.

G

GEMCOS

See Generalized Message Control System.

Generalized Message Control System (GEMCOS)

A message control system (MCS) developed for online systems. GEMCOS is transaction oriented.

GIVING clause

In COBOL, an optional clause in the *CALL nonnumeric literal* statement that specifies a variable as the destination for the integer value returned from the procedure call.

global data item

A data item, group item, or population item that is not a part of any data set. Global data items generally contain information such as control totals, hash totals, and populations that apply to the entire database.

group item

A collection of data items that can be viewed as a single data item.

H

halt/load

A system-initialization procedure that temporarily halts the system and loads the operating system from a disk to main memory.

hashing algorithm

An algorithm used in constructing and maintaining hash tables, which modify record keys to produce the addresses of the record keys in a structure.

I

ID

See identifier.

identifier (ID)

(1) A label. (2) One node of a file name.

import

To bring in from an outside source and still retain the original significance or function.

IN LIBRARY statement

In FORTRAN77, the statement that associates the logical expression name of an entry point with the library that is known to the application program through the library identifier.

INCLUDE

A compiler control option that enables the user to insert an entire file, or sequence range within a file, into a program.

index sequential structure

A structure that is a collection of tables arranged hierarchically.

input/output mapping clause

A clause that can be used with data management operations to assign the value of a database item to a program variable (input) and/or to assign the value of a program variable to a database item (output).

inquiry mode

A database mode in which data can be read but not updated.

integer

A whole number.

internal file name

The name used to declare a logical file in a program. The internal name of a file is given by the value of its INTNAME file attribute. Work Flow Language (WFL) file equation statements can reference the file by implicitly or explicitly specifying an INTNAME value that matches the INTNAME attribute of a file in a program.

interpretive interface

A set of conventions for passing information using the DMINTERPRETER library.

INTNAME

See internal file name.

intrinsic

A system-supplied program routine for common mathematical and other operations that is loaded onto the system separately. An intrinsic can be invoked by the Master Control Program (MCP), by user programs, or by the system operator.

intrinsic data set

A data set created by system designers as part of a Data Management System II (DMSII) software package, as opposed to a data set created by a user.

invocation

(1) The syntax used to initiate execution of software. (2) The act that transfers control to the start of a specified procedure, initializes any parameters, and begins the execution of the statements of the procedure. Invocations are of two kinds: entrances and initiations.

invoke

(1) To cause to be executed. (2) To cause to be brought into main storage.

item

A field in a database record that contains an individual piece of information and can be referenced by name.

J**job**

A group of one or more tasks under the control of a single Work Flow Language (WFL) program. The system assigns each job a mix number and treats each job as a discrete unit of work. See WFL job.

K

key

(1) A field used to locate or identify a record in an indexed file. (2) A field in a record that is used to sort a file. (3) *See also* key item.

key field

See key.

key item

A data item or group item that serves as a retrieval key for a set, subset, or access.

L

language extension

A syntactical addition to a standard language compiler that enables the compiler to interface directly with Unisys products.

library

(1) A collection of one or more named routines or library objects that are stored in a file and can be accessed by other programs. (2) A program that exports objects for use by user programs.

library directory

A memory structure associated with a library process stack that describes the objects exported by the library process.

library identifier

An internal name within an application program that represents the library code file being accessed.

linear search

See serial search.

link item

A field that enables one data set record to refer to another.

logical database

A collection of structures declared in the Data and Structure Definition Language (DASDL) that provide a view of the database, enforce structure-level security, and achieve data independence. When a logical database is declared in the DASDL, the data sets, sets, subsets, and remaps to be included in it are listed.

logical expression

In FORTRAN77, a rule for computing a value corresponding to *.TRUE.* or *.FALSE.* . Logical expressions consist of any valid combination of logical operands, logical operators, and parentheses.

logical function

In FORTRAN, a function that returns a value of *.TRUE.* or *.FALSE.* .

M

major key

The first key in a complex key.

manual subset

A subset that has no condition specifying which data set records are to be included in the subset. The user must add and delete manual subset entries, using the INSERT and REMOVE statements.

master

A data set or record that contains one or more embedded data sets or records. In DMSII, *synonym for master, parent, owner.*

Master Control Program (MCP)

An operating system on A Series systems. The MCP controls the operational environment of the system by performing job selection, memory management, peripheral management, virtual memory management, dynamic subroutine linkage, and logging of errors and system utilization.

MCP

See Master Control Program.

MCS

See message control system.

member

A record of a data set.

message control system (MCS)

A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

metatoken

An item that appears in syntax notation as a variable item.

minor key

Any key in a complex key that is not the major (first) key.

mnemonic

(1) An abbreviation or acronym that is used to assist the human memory. (2) A programmer-supplied word associated with a specific function name. (3) A character or group of characters intended to serve as a mnemonic.

N

nonkey item

An item in the data set being indexed that is not a key item in the selected set.

null string

An empty or zero-length string.

null value

The value contained in an item that does not contain valid information.

numeric item

A data item whose description restricts its contents to a value represented by numeric characters.

O

object code file

A file produced by a compiler when a program is compiled successfully. The file contains instructions in machine-executable object code.

OCCURS clause

In a Data and Structure Definition Language (DASDL) data item declaration, a clause that establishes an ordered (subscripted) collection of data items with identical attributes.

OCM

See Operations Control Manager.

Operations Control Manager (OCM)

A menu-driven member of the InfoExec family of products that is used to maintain databases.

ordered

Pertaining to an item maintained in a user-specified sequence.

owner

See master.

P

parent

A process that owns the critical block of a dependent process. If the parent exits the critical block before the dependent process terminates, the dependent process is discontinued. *See also* master.

path

(1) The route that must be traced from a directory to a subdirectory, or through a series of subdirectories, to find a file. (2) A specific location within the logical ordering of a data set, set, subset, or access.

physical database

An entire database as it is stored on a disk. Whereas a logical database represents only parts of a database to be used for limited purposes, a physical database is an entire database.

population

For disjoint data sets, the number of records in the data set. For embedded data sets, the population is the number of records in the embedded data set owned by the current master.

Q**qualification**

The specification of the data set that owns an item. Qualification is usually used when several data sets contain an item with the same name.

quiet point

A time during which there are no transactions in progress for a database. *See also* syncpoint.

R**railroad diagram**

A graphic representation of the syntax of a command or statement.

RDS

See restart data set.

real item

A data item that stores signed or unsigned, fractional or whole values in single-precision, floating-point form.

real number

Any number, including fractions and whole numbers.

rebuild

In database management, a recovery process in which the entire database is loaded from one or more sets of dump tapes. The recovery process then applies the audit trail after-update record images to move the database forward in time.

record

(1) A group of logically related items of data in a file that are treated as a unit. (2) The data read from or written to a file in one execution of a read or write statement in a program.

record-type item

Either of the following: a binary integer value used in conjunction with variable-format data sets to identify the variable-format part, if any, the record contains; or a control item that contains the format number for records in variable-format data sets.

recovery

In data management, a procedure that is initiated following a hardware, software, or operations failure while the database is in update mode. Recovery backs out any partially completed transactions by applying audit-trail images to the database to restore

it to a consistent state. In addition, recovery passes restart information to the programs accessing the database.

remap

A logical data set that redefines a physical data set by omitting, reordering, or renaming the items.

remote file

A file with the KIND attribute specified as REMOTE. A remote file enables object programs to communicate interactively with a terminal.

reorganization

The process of reordering or reformatting data sets, sets, or subsets. Reorganization can restore space in files, reorder data sets for more efficient retrieval, and reformat data set records when items are added, deleted, or changed.

restart data set (RDS)

A data set containing restart records that application programs can access to recover database information after a system failure.

restart record

A record containing user-defined information that enables a user program to restart in response to a particular condition.

result descriptor

A 48-bit word that is the standard Data Management System II (DMSII) exception word. A result descriptor includes the exception category and subcategory, and the structure number involved. When an exception occurs, bit 0 in the 48-bit word is set to 1. When an operation is successful, the 48-bit word contains all zeros.

rollback

The recovery of a database or transaction base to a consistent state at an earlier point in time.

RPG

Report Program Generator. A high-level, commercially oriented programming language used most frequently to produce reports based on information derived from data files.

run time

The time during which an object code file or user interface system (UIS) is executed. *Synonym for execution time and, in COBOL, object time.*

run-time error

An error occurring during the execution of a program, which causes the system software to terminate execution of that program abnormally.

S

savepoint

An user-determined point in a program that is located between a begin transaction marker and its corresponding end transaction marker. While executing the transaction, processing can be specifically backed out to this point.

schema

The outline or description of a database. The schema acts as a map for the host system to use when performing any functions on the database or when accessing the database.

selection expression

The entire complement of selection criteria used in a search operation to locate a data set record. The definition of a selection expression encompasses both the select options (FIRST, NEXT, LAST, and PRIOR) and all the variations for the key conditions. *See also* condition.

Semantic Information Manager (SIM)

The basis of the InfoExec environment. SIM is a database management system used to describe and maintain associations among data by means of subclass-superclass relationships and linking attributes.

serial search

A method for searching a set of information by examining each element of the set one at a time.

set

A file of indexes that refers to all the records of a single data set. Sets are automatically maintained by the system. Sets permit access to the records of a data set in some logical sequence and are normally used to optimize certain types of retrievals of the data set records.

SIM

See Semantic Information Manager.

source file

(1) A file in which a source program is stored. (2) A file containing instructions written in a programming language.

standard entry point

In the DMINTERPRETER library, an entry point that performs a function corresponding to a function specified in a Data Management System II (DMSII) user language statement.

string

A connected sequence or group of characters.

structure

A data set, set, subset, access, or remap.

subitem

An item that is a member of a group item.

subscript

A number that is an index into an array.

subset

An index structure that is identical to a set, except that the subset need not contain a record for every record of the data set. A set must index every record in its associated data set, whereas a subset can index zero, one, several, or all data set records. A subset might or might not be automatically maintained by Data Management System II (DMSII).

symbolic name

A basic syntactic item of the FORTRAN77 language. A symbolic name is a sequence of from 1 to 31 letters, digits, underscores (_), or the dollar sign symbol (\$). The first character must be either a letter or the dollar sign symbol.

syncpoint

A point in time when no program is in a transaction state.

syntax

The rules or grammar of a language.

T

TFL

See Transaction Formatting Language.

timestamp

An encoded, 48-bit numerical value for the time and date. Various timestamps are maintained by the system for each disk file. Timestamps note the time and date a file was created, last altered, and last accessed.

TITLE

A file attribute whose value is the external name of a file. By default, this value is the value of the INTNAME attribute. For a file whose KIND attribute is equal to DISK or PACK, the TITLE attribute can be assigned a value of the form *<file name> ON <family name>*; thus, the values of the TITLE and FILENAME attributes, both of which specify the external file name, can be different.

TPS

See transaction processing system.

transaction

(1) The transfer of one message from a terminal or host program to a receiving host program, the processing carried out by the receiving host program, and the return of an answer to the sender. (2) In data management, a sequence of operations grouped by a user program because the operations constitute a single logical change to the database.

Transaction Formatting Language (TFL)

The Unisys language used to write source files that are compiled to produce description files for transaction bases.

transaction point

A point that is explicitly assigned in a program between a begin transaction statement and an end transaction statement so that the programmer is able to cancel or partially cancel a transaction that has not yet completed processing.

transaction processing system (TPS)

A Unisys system that provides methods for processing a high volume of transactions, keeps track of all input transactions that access the database, enables the user to batch data for later processing, and enables transactions to be processed on a database that resides on a remote system.

transaction state

The period in an application language program between a begin transaction operation and an end transaction operation.

U**unordered**

Referring to files, data sets, sets, and subsets that are not maintained in a user-specified order.

update

To delete, insert, or modify information in a database or transaction base.

update mode

A database or file access mode in which data can be inserted, deleted, or modified.

user work area

A memory area in a user program where data records are constructed, accessed, or modified. The Accessroutines maintain one user work area for each data set or remap invoked by a program.

USING clause

In COBOL, an optional clause in the *CALL nonnumeric literal* statement that identifies the parameters being passed to a library entry point.

utility

A system software program that performs commonly used functions.

V**variable**

An object in a program whose value can be changed during program execution.

variable format

A record format that consists of two parts: a fixed part and a variable-format part. A single record description exists for the fixed part. The variable-format part can describe several variable parts. An individual record is constructed by using the fixed part alone, or by joining the fixed part with one of the variable parts.

variable-format data set

A data set with records that can have several different formats. All records in such a data set, known as variable-format records, consist of a common fixed part and optionally include one of the variable parts declared in the Data and Structure Definition Language (DASDL).

W

WFL

See Work Flow Language.

WFL job

(1) A Work Flow Language (WFL) program, or the execution of such a program. (2) A collection of Work Flow Language (WFL) statements that enable the user to run programs or tasks.

WFL job deck

See job.

word

A unit of computer memory. On A Series systems, a word consists of 48 bits used for storage plus tag bits used to indicate how the word is interpreted.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

Z

ZIP

A statement that initiates the Work Flow Language (WFL) compiler. A ZIP is commonly used to initiate compilations automatically.

Bibliography

- A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation* (form 1169844). Unisys Corporation.
- A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces* (form 1170099). Unisys Corporation.
- A Series ALGOL Test and Debug System (TADS) Programming Guide* (form 1169539). Unisys Corporation.
- A Series C Programming Reference Manual* (form 1221054). Unisys Corporation.
- A Series CANDE Configuration Reference Manual* (form 1170065). Unisys Corporation.
- A Series CANDE Operations Reference Manual* (form 1169869). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* (form 1169877). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces* (form 1180619). Unisys Corporation.
- A Series COBOL ANSI-74 Test and Debug System (TADS) Programming Guide* (form 1169901). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation* (form 5031503). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 2: Product Interfaces* (form 5044308). Unisys Corporation.
- A Series COBOL Programming Reference Manual* (form 1169786). Unisys Corporation.
- A Series DMALGOL Programming Reference Manual* (form 1170248). Unisys Corporation.
- A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* (form 1163805). Unisys Corporation.
- A Series DMSII Data Dictionary Reference Manual* (form 1177128). Unisys Corporation.
- A Series DMSII DataAid User's Guide* (form 1180544). Unisys Corporation.
- A Series DMSII Inquiry Operations Guide* (form 1164035). Unisys Corporation.
- A Series DMSII Interpretive Interface Programming Reference Manual* (form 1163813). Unisys Corporation.

- A Series DMSII Technical Overview* (form 5044191). Unisys Corporation.
- A Series DMSII Transaction Processing System (TPS) Programming Guide* (form 1164043). Unisys Corporation.
- A Series DMSII Utilities Operations Guide* (form 1163839). Unisys Corporation.
- A Series Documentation Library Overview* (form 1170156). Unisys Corporation.
- A Series Editor Operations Guide* (form 1169976). Unisys Corporation.
- A Series Extended Retrieval with Graphic Output (ERGO) Operations Guide* (form 1164027). Unisys Corporation.
- A Series FORTRAN77 Programming Reference Manual* (form 1222666). Unisys Corporation.
- A Series FORTRAN77 Test and Debug System (TADS) Programming Guide* (form 1222667). Unisys Corporation.
- A Series InfoExec Administration Guide* (form 1195070). Unisys Corporation.
- A Series InfoExec Capabilities Manual* (form 1195088). Unisys Corporation.
- A Series InfoExec Functional Overview* (form 1195138). Unisys Corporation.
- A Series InfoExec Operations Guide* (form 1195120). Unisys Corporation.
- A Series Mark 3.8 Software Release Capabilities Overview* (form 5044118). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation* (form 1169851). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 2: Product Interfaces* (form 1170107). Unisys Corporation.
- A Series PL/I Reference Manual* (form 1169620). Unisys Corporation.
- A Series Railroad Diagram Reference Card* (form 5044266). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Reference Manual, Volume 1: Basic Implementation* (form 1169760). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Reference Manual, Volume 2: Product Interfaces* (form 1170115). Unisys Corporation.
- A Series Report Program Generator (RPG) Programming Template* (form 1195302). Unisys Corporation.
- A Series Security Administration Guide* (form 1195195). Unisys Corporation.

A Series Security Features Operations and Programming Guide (form 1195203). Unisys Corporation.

A Series Software Documents Master Index (form 1170198). Unisys Corporation.

A Series System Administration Guide (form 5044068). Unisys Corporation.

A Series Systems Functional Overview (form 1170057). Unisys Corporation.

A Series Work Flow Administration and Programming Guide (form 1170149). Unisys Corporation.

A Series Work Flow Language (WFL) Programming Reference Manual (form 1169802). Unisys Corporation.

The DMSII Primer, Volume 1. Sonoma, CA: Gregory Publishing Company, 1982.

Vocabulary for Data Processing, Telecommunications, and Office Systems (form GC10-1699-6). Poughkeepsie, New York: International Business Machines Corporation, 1981.

Index

A

ABORT

- major category number, A-2

ABORT task, 8-1

- conventional algorithm, 8-2
- exception and error subcategories, A-3
- when initiated, 8-1
- where BACKOUTTOSPT is set, 8-2
- where INDEPENDENTTRANS is set, 8-5

abort transaction operations

- effect of INDEPENDENTTRANS, 8-12

aborting transactions, using language

- extensions, 8-12

ABORTTRANSACTION, (See abort transaction operations)

- language extension, 3-8

access modes

- inquiry, 4-4
- update, 4-4

accessing databases

- invoking and declaring, 4-1
- opening, 4-4
- using parameterized open operations, 1-5

accessing user work areas, through entry points, 1-2

Accessroutines

- as a run-time interface
 - for the interpretive interface, 1-7
 - for the language extensions, 1-4
- exceptions and errors returned, A-1
- handling fatal errors, 9-2
- handling nonfatal errors, 9-2
- initiating the ABORT task, 8-1
- initiating the DMRECOVERY program, 8-2
- maintaining audit trails, 8-4
- performing readaheads, 6-22
- reentrance capability, 1-7
- tasks, 1-7
- using syncpoints, 8-10

adding and deleting restart data set records, 8-7

adding key entries in manual subsets, 7-5

adding new records, 7-2

ADDS, (See Advanced Data Dictionary System (ADDS))

Advanced Data Dictionary System (ADDS)

- using to define databases, 2-2

advantages of

- interpretive interface, 1-5
- language extensions, 1-2

afterimages

- definition of, 2-5
- stored in audit trails, 2-7

ALGOL

- sample interpretive interface program, B-5
- sample language extension program, C-7

analyzing the structure of databases, 2-3

AND operator

- combining with NOT and OR operators, 6-16
- syntax of, 6-7
- using in selection expressions, 6-14

approximate matches, coding selection expression, 6-12

arithmetic symbols for relational operators, 6-10

ascending order

- for key items, 6-9
- use in search operations, 6-12

ASSIGN, language extension, 3-8

audit file, in the recovery process, 8-3

AUDIT option

- defining audited databases in DASDL, 8-3
- producing audit trails, 8-4
- using in begin and end transaction operations, 8-7, 8-10
- writing records to the restart data set, 8-7, 8-10

audit trails

- purpose of, 8-1
- recovering audited databases, 2-7, 8-4
- storing beforeimages and afterimages, 2-7
- audited databases**
 - changing the values of required items, 7-5
 - creating new data set records, 7-3
 - defining in DASDL, 8-3
 - differences from unaudited databases, 2-5
 - invoking the database, 4-1
 - minimum requirements, 8-3
 - open operations, 4-4
 - processing transactions, 2-5
 - recovering
 - overview, 2-7
 - programming guidelines, 8-1
 - updating records, 7-4
 - using audit trails, 2-7, 8-4
 - using data management operations, 8-8
- AUDITERORR**
 - exception and error subcategories, A-3
 - major category number, A-2
- automatic subsets**
 - creating entries, 7-3, 7-5
 - deleting entries, 7-5
- automatically restoring databases, 8-4

B

- backing out transactions, 8-5
- BACKOUTTOSPT** option
 - defining audited databases in DASDL, 8-3
 - effect on ABORT task, 8-2
- backward serial searches, 6-4
- beforeimages**
 - definition of, 2-5
 - stored in audit trails, 2-7
- begin transaction operations, 8-9
 - purpose of, 2-6
 - using the AUDIT option, 8-10
- beginning of a structure, 2-9
- BEGINTRANSACTION**, language extension, 3-8
- binary searches**, 6-18
 - combining with serial searches, 6-25
 - complex key, 6-25
 - using the AND operator, 6-14
- Boolean operators**
 - truth table, 6-13
 - using the AND operator, 6-14
 - using the AND, OR, and NOT operators, 6-16

- using the NOT operator, 6-13
- using the OR operator, 6-15

C

- CANCEL TRANSACTION POINT**, (See cancel transaction point operations)
 - language extension, 3-8
- cancel transaction point operations
 - effect of INDEPENDENTTRANS, 8-12
- canceling transactions, using language extensions, 8-12
- CANDE**, (See Command and Edit (CANDE))
- categories of exceptions and errors, A-2
- certifying the integrity of databases, 2-3
- changing nonrequired items, 7-3
- changing required items, 7-4
- changing the value of items, 7-3
- close operations
 - effect on databases, 4-5
 - freeing structures, 2-15
- CLOSE**, language extension, 3-8
- CLOSEERROR**
 - exception and error subcategories, A-4
 - major category number, A-2
- closing databases, general guidelines, 4-5
- COBOL** example of a restartable program, 8-13
- COBOL74**
 - sample interpretive interface program, B-15
 - sample language extension program, C-16
- combining serial and random search operations, 6-17
- Command and Edit (CANDE)**
 - database equation, 4-3
- commands, (See language extensions)
- Communications Management System (COMS)**
 - sample program for COBOL74, C-16
 - sample program for RPG, C-26
- compile-time interfaces**
 - with the interpretive interface, 1-7
 - with the language extensions, 1-4
- complex key items, 6-8
- complex key, performing a binary search on, 6-25
- < complex selection expression >, 6-7
- COMPUTE**, language extension, 3-8

COMS, (See Communications Management System (COMS))

concurrency control, enforced by
 INDEPENDENTTRANS, 8-5

control system resources for databases, 2-3

create operations

- adding new records, 7-3
- difference from recreate operations, 2-15
- setting the current path, 5-1
- using to write records to the restart data set, 8-10

CREATE, language extension, 3-8

creating selection expressions, 6-7

current path

- defining status of, 2-12
- definition of, 2-8
- for data sets, 2-12
- for sets or subsets, 2-12
- path adjustment rules for record deletions, 5-5
- setting by data management operations, 5-1
- setting in a find operation
 - for sets or subsets, 5-3
- setting in find operations
 - for data sets, 5-2
- setting to the beginning of the structure, 5-1
- setting to the end of the structure, 5-1
- setting to the first record, 6-5
- setting to the last record, 6-6
- setting to the next record, 6-5
- setting to the prior record, 6-6
- using virtual records, 2-12

current record

- defining status of, 2-13
- definition of, 2-8
- states of, 2-13

D

DASDL, (See Data Structure and Definition Language (DASDL))

data management commands, (See language extensions)

data management operations

- for audited databases, 3-3
- for unaudited databases, 3-2
- in interpretive interface programs, 3-3
- in language extension programs, 3-7
- in transaction state, 3-2, 8-8

data management statements, (See language extensions)

data management systems, goal, 2-1

data management tasks, overview, 3-1

data sets

- finding through selection expressions, 6-25
- resetting an invalid current path, 5-2
- valid search types and techniques, 6-27

Data Structure and Definition Language (DASDL)

- database for sample interpretive interface programs, B-2
- declaring a restart data set, 8-13
- defining audited databases, 8-3
- defining databases, 2-2
- for ALGOL sample language extension program, C-2
- for RPG sample language extension programs, C-25
- table of recovery options, 8-3
- use of options
 - AUDIT, 8-3, 8-4
 - BACKOUTTOSPT, 8-3
 - INDEPENDENTTRANS, 2-15, 8-3, 8-5, 8-9, 8-12
 - RDSSTORE, 8-3, 8-10
 - REAPPLYCOMPLETED, 8-3, 8-5
 - RESTART TRANSACTION ONLY, 8-7

data transfer entry points, table of, 3-5

data transfer operations

- adding new records, 7-3
- changing nonrequired items, 7-3
- changing required items, 7-4

data validation, definition of, 2-2

database declaration, sharing in multiple tasks, 4-4

database definition for

- ALGOL sample language extension program, C-2
- RPG sample language extension program, C-25
- sample interpretive interface programs, B-2

database equation, overriding a database title, 4-2

< database equation >, WFL syntax, 4-2

database schema

- changing, 2-3
- creating, 2-2
- definition of, 2-1

databases

- audited, 2-5

- closing, 4-5
- declaring, 4-1
- defining schema, 2-2
- difference between audited and unaudited, 2-5
- difference between invoking and opening, 2-4
- invoking, 4-1
- monitoring operations, 2-3
- multitasking, 8-11
- opening, 4-4
- processing updates for audited and unaudited, 2-5
- processing with the interpretive interface, 1-5
- processing with the language extensions, 1-2
- recovering, 2-5
- restoring audited, 8-4
- restoring unaudited, 2-7
- unaudited, 2-5
- when in transaction state, 2-6
- DATAERROR**
 - exception and error subcategories, A-4
 - major category number, A-2
- DBANALYZER**, analyzing the structure of databases, 2-3
- DBCERTIFICATION**, certifying the integrity of databases, 2-3
- DBEXCEPTIONNAME**, returning the name of an exception, 9-4
- DBEXCEPTIONTEXT**, returning the text of an exception message, 9-4
- DBSTATUS** exception word, 9-4
- DBSTRUCTURE**, interrogating the structure of databases, 1-5
- DEADLOCK**
 - exception and error subcategories, A-4
 - major category number, A-2
- deadlocks
 - exceptions and errors, A-4
 - involving nondata management operations, 8-11
 - preventing, 7-2
 - recovering from, 7-2
- deadly embrace, (See deadlocks)
- declaring databases, 4-1
- defined current record, status of current record, 2-13
- defining audited databases, 8-3
- defining database structures, 2-2
- defining databases, 2-1
- delete operations
 - as a search and retrieval operation, 6-1
 - changing values of required items, 7-4
 - for records, 2-13
 - setting the current path, 5-1
 - using readaheads, 6-22
 - using reblocking, 6-22
- DELETE**, language extension, 3-8
- deleting key entries from manual subsets, 7-5
- deleting records
 - effect on program paths, 5-5
 - guidelines, 7-5
- deleting records from the restart data set, 8-7
- descending order
 - for key items, 6-9
 - use in search operations, 6-13
- description file
 - for database structures, 2-3
 - use when invoking databases, 1-4
- DESCRIPTION**/**< database name >**, use when invoking databases, 1-4
- DMCATEGORY** field, 9-4
- DMERROR** field, 9-4
- DMERRORTYPE** field, 9-4
- DMINTERFACE**, compile-time interface, 1-4
- DMKEY**, language extension, 3-8
- DMMONITOR**, monitoring database status, 2-3
- DMRECOVERY** program
 - recovering from a halt/load, 8-2
- DMRESULT** operation, 9-4
- DMSAVEPOINT**, (See savepoint operations)
- language extension, 3-8
- DMSTATUS** register, 9-4
- DMSTRUCTURE** field, 9-4
- DMTERMINATE**, (See terminate operations)
- language extension, 3-8
- DMUTILITY**, monitoring databases, 2-3
- dumping
 - for audited databases, 8-1
 - for unaudited databases, 2-8
- duplicate keys, using in selection expressions, 6-9
- DUPLICATES**
 - exception and error subcategories, A-5
 - major category number, A-2

E

- embedded manual subsets, with the
 - interpretive interface, 1-2
- end of a structure, 2-9
- end transaction operations, 8-9
 - concurrency control with
 - INDEPENDENTTRANS, 8-5
 - purpose of, 2-6
 - using syncpoints, 8-11
 - using the AUDIT option, 8-10
- ENDTRANSACTION, language extension, 3-8
- enhancements to the find operation, 6-24
- entry points
 - table of data transfer entry points, 3-5
 - table of exception handling entry points, 3-7
 - table of standard entry points, 3-4
- EQU operator, (See equal to operator)
- equal to operator, 6-10
 - using in selection expressions, 6-11
- equal to or greater than operator, 6-10
 - use in selection expressions, 6-12
- equal to or less than operator, 6-10
 - using in selection expressions, 6-12
- error condition, definition, 9-1
- errors
 - fatal, 9-1
 - major categories by number, A-2
 - masking, 9-3
 - nonfatal, 9-1
 - subcategories, A-2
- evaluating selection expressions, 6-24
- exact matches, coding selection expressions, 6-11
- examples
 - coding restartable programs, 8-12
 - locating data set records, 6-22
 - logical space between two existing records, 2-13
 - using a WFL database equation, 4-2
 - using a WFL database equation through CANDE, 4-3
 - using AND operators, 6-14
 - using AND, OR, and NOT operators together, 6-16
 - using binary search techniques, 6-19
 - using OR operators, 6-16
 - using relational operators, 6-11
- exception condition, definition, 9-1
- exception handling, 9-1

- exception handling entry points, table of, 3-7

- exceptions
 - fatal, 9-1
 - major categories by number, A-2
 - masking, 9-3
 - nonfatal, 9-1
 - returning NOTFOUND as outcome of a search, 6-3
 - subcategories, A-2

F

- fatal errors and exceptions
 - definition of, 9-1
 - detecting, 9-1, 9-2
 - during the DMRECOVERY program, 8-3
- FATALERROR
 - exception and error subcategories, A-5
 - major category number, A-2
- find first operations
 - in combined searches, 6-17
 - retrieving the first physical record, 6-5
- find key of operations, 6-18
 - in the interpretive interface, 6-26
- find last operations
 - in combined searches, 6-17
 - retrieving the last physical record, 6-6
- find next operations
 - in combined searches, 6-17
 - retrieving the next physical record, 6-5
 - setting an invalid current path, 5-3
- find operations
 - as a search and retrieval operation, 6-1
 - difference from lock or secure operations, 2-13, 7-1
 - enhancements for the interpretive interface, 6-24
 - for records, 2-13
 - initiating partial binary searches, 6-20
 - retrieving the current record, 6-4
 - setting the current path, 5-1
 - for a set or subset, 5-3
 - for data sets, 5-2
 - using readaheads, 6-22
 - using reblocking, 6-22
- find prior operations
 - in combined searches, 6-17
 - retrieving the previous physical record, 6-6
 - setting an invalid current path, 5-3
- FIND, language extension, 3-8
- finding records in a set or subset, 6-18

first physical record, retrieving, 6-5
 FORTRAN77 sample interpretive interface
 program, B-25
 forward serial searches, 6-3
 free operations
 for records, 2-14
 for structures, 2-15
 FREE, language extension, 3-8
 freeing records
 difference between find and secure
 operations, 7-1
 reasons for, 2-13, 2-14
 freeing structures, 2-15

G

GENERATE, language extension, 3-8
 GEQ operator, (See equal to or greater than operator)
 GET, language extension, 3-8
 greater than operator, 6-10
 using in selection expressions, 6-12
 GTR operator, (See greater than operator)
 guidelines
 for search and retrieval operations, 6-27
 processing transactions for audited
 databases, 2-6
 using the AND operator in selection
 expressions, 6-14
 using the OR operator in selection
 expressions, 6-16

H

halt/load
 freeing structures, 2-15
 recovery from, 8-2
 when initiated, 8-1
 when REAPPLYCOMPLETED is set, 8-5
 handling exceptions and errors
 fatal errors, 9-2
 in an application program, 9-4
 nonfatal errors, 9-2
 through DMSII software, 9-3

I

IF, language extension, 3-8

INDEPENDENTTRANS option
 defining audited databases in DASDL, 8-3
 effect in halt/load recovery, 8-3
 effect on aborting transactions, 8-12
 effect on canceling transactions, 8-12
 effect on end transaction operations, 8-9
 effect on recovery process, 8-5
 securing structures, 2-15
 use with REAPPLYCOMPLETED, 8-5
 inquiry access mode, 4-4
 insert operations
 inserting key entries in manual subsets,
 7-5
 setting the current path, 5-1
 INSERT, language extension, 3-8
 INTEGRITYERROR
 exception and error subcategories, A-5
 handling nonfatal errors, 9-2
 major category number, A-2
 interpretive interface
 accessing multiple databases, 1-5
 advantages over the language extensions,
 1-5
 basics of, 1-4
 compile-time interface, 1-7
 data management entry points, 3-3
 entry points, table of, 3-4
 find operation enhancements, 6-24
 opening databases, 4-4
 run-time interface, 1-7
 sample programs
 ALGOL, B-5
 COBOL74, B-15
 DASDL source for, B-2
 FORTRAN77, B-25
 using parameterized open operations, 1-5
 using the DBSTRUCTURE data set, 1-5
 interrogating the structure of databases at
 run time, 1-5
 INTLIBERROR
 exception and error subcategories, A-6
 major category number, A-2
 INUSE
 exception and error subcategories, A-7
 major category number, A-2
 invalid current path
 for data sets, 2-12
 for sets or subsets, 2-12
 invoking databases, 4-1
 difference from opening databases, 2-4
 using the description file, 1-4
 IOERROR

exception and error subcategories, A-8
major category number, A-2

K

key, (See key items)
key data, definition of, 6-18
key entries
 inserting in manual subsets, 7-5
 removing from manual subsets, 7-5
key items
 ascending and descending order, 6-9
 complex, 6-8
 definition of, 6-8
 simple, 6-8
 using in search operations, 6-20
 using in selection expressions, 6-8, 6-10
key, definition of, 2-12
KEYCHANGED
 exception and error subcategories, A-9
 major category number, A-2

L

language extensions
 ABORTTRANSACTION, 3-8, (See also abort transaction operations)
 advantages over the interpretive interface, 1-2
 ASSIGN, 3-8
 basics of, 1-2
 BEGINTRANSACTION, 3-8, (See also begin transaction operations)
 CANCEL TRANSACTION POINT, 3-8, (See also cancel transaction point operations)
 CLOSE, 3-8, (See also close operations)
 comparing to the interpretive interface, 1-2
 compile-time interface, 1-4
 COMPUTE, 3-8
 CREATE, 3-8, (See also create operations)
 data management operations, 3-7
 data management statements and commands, 3-8
 DELETE, 3-8, (See also delete operations)
 DMKEY, 3-8

DMSAVEPOINT, 3-8, (See also savepoint operations)
DMTERMINATE, 3-8, (See also terminate operations)
effect of INDEPENDENTTRANS, 8-12
ENDTRANSACTION, 3-8, (See also end transaction operations)
FIND, 3-8, (See also find operations)
FREE, 3-8, (See also free operations)
GENERATE, 3-8
GET, 3-8
IF, 3-8
INSERT, 3-8, (See also insert operations)
LOCK, 3-8, (See also lock operations)
MODIFY, 3-8, (See also lock operations)
OPEN, 3-9, (See also open operations)
opening databases, 4-4
PUT, 3-9
recovery statements and commands, 8-11
RECREATE, 3-9, (See also recreate operations)
REMOVE, 3-9, (See also remove operations)
run-time interface, 1-4
sample programs
 ALGOL, C-7
 COBOL74, C-16
 RPG, C-26
SECURE, 3-9, (See also secure operations)
SET, 3-9, (See also set operations)
STORE, 3-9, (See also store operations)
last physical record, retrieving, 6-6
LEQ operator, (See equal to or less than operator)
less than operator, 6-10
 using in selection expressions, 6-12
libraries, use in updating audited databases, 2-6
LIMITERROR
 exception and error subcategories, A-9
 major category number, A-2
linear searches, (See serial searches)
link item, using in selection expressions, 6-8
lock operations
 as a search and retrieval operation, 6-1
 changing nonrequired items, 7-3
 concurrency control with INDEPENDENTTRANS, 8-5
 difference from find or secure operations, 2-13, 7-1
 for records, 2-13

- for structures, 2-15
- setting the current path, 5-1
- using readaheads, 6-22
- using reblocking, 6-22
- LOCK, language extension, 3-8
- locking records, 7-2
 - preventing deadlocks, 7-2
 - reasons for, 2-13
- locking structures, 2-15
- logical databases
 - invoking, 4-1
 - using remaps, 4-1
- logical transactions, 2-6
- LSS operator, (See less than operator)

M

- maintaining links, with the interpretive interface, 1-2
- major categories for exceptions and errors, table of, A-2
- major keys, 6-8
- manipulating data values, 2-4
- manual recovery, of unaudited databases, 2-8
- manual subsets
 - inserting key entries, 7-5
 - removing key entries, 7-5
- manually restoring databases, 2-7
- masking messages, 9-1
- matching values
 - approximately, 6-12
 - exact alphanumeric matches, 6-12
 - exact numeric matches, 6-12
- minor keys, 6-8
- modify operations, (See lock operations)
 - as a search and retrieval operation, 6-1
 - for records, 2-13
 - setting the current path, 5-1
 - using readaheads, 6-22
 - using reblocking, 6-22
- MODIFY, language extension, 3-8
- modifying records, 7-2
- monitoring the operation of databases, 2-3
- monitoring the status of databases, 2-3
- multitasking, 8-11

N

- NEQ operator, (See not equal to operator)

- next physical record, retrieving, 6-5
- nonexistent records, 2-12
- nonfatal errors and exceptions
 - definition of, 9-1
 - detecting, 9-1, 9-2
- nonkey items, using in selection expressions, 6-10
- nonrequired items, changing, 7-3
- NORECORD
 - exception and error subcategories, A-10
 - major category number, A-2
- not equal to operator, 6-10
- NOT operator
 - combining with AND and OR operators, 6-16
 - syntax of, 6-7
 - using in selection expressions, 6-13
- NOTFOUND
 - exception and error subcategories, A-10
 - major category number, A-2
 - preventing as outcome of a search, 6-3
- NOTLOCKED
 - exception and error subcategories, A-11
 - major category number, A-2

O

- OCM, (See Operations Control Manager (OCM))
- open operations
 - retrieving records, 7-1, 7-2
 - using parameters, 1-5
- OPEN, language extension, 3-9
- OPENERROR
 - exception and error subcategories, A-11
 - major category number, A-2
- opening databases, 4-4
 - difference from invoking databases, 2-4
 - general description, 4-4
 - position of current paths, 2-9
- operations
 - aborting transactions, 8-12
 - adding key entries in automatic subsets, 7-3, 7-5
 - adding key entries in manual subsets, 7-5
 - beginning transactions, 2-6
 - canceling transactions, 8-12
 - changing nonrequired items, 7-3
 - changing required items, 7-4
 - closing databases, 4-5
 - creating new records, 7-3

- creating records, 2-15
- deleting data set records, 7-5
- deleting key entries from manual subsets, 7-5
- deleting key entries in automatic subsets, 7-5
- deleting records, 2-13
- ending transactions, 2-6
- find key of, 6-18
- freeing records, 2-13, 2-14
- freeing structures, 2-15
- in transaction state, 8-8
- inserting key entries in manual subsets, 7-5
- invoking a database, 4-1
- locking records, 2-13, 7-2
- locking structures, 2-15
- modifying records, 2-13, 7-2
- opening databases, 4-4
- performed only in transaction state, 2-6
- re-creating records, 2-15
- reblocking, 6-22
- removing key entries from manual subsets, 7-5
- searching for and retrieving records, 6-1
 - by selection criteria, 6-7, 6-17
 - randomly, 6-7
 - randomly and serially, 6-17
 - serially, 6-3
- securing records, 2-13, 7-2
- securing structures, 2-15
- setting a savepoint, 8-12
- setting the current path, 5-1
- storing new records, 7-3
- terminating a program, 8-12
- using concurrency control, 8-5
- Operations Control Manager (OCM)
 - monitoring database operations, 2-3
- options, use of DASDL, (See Data Structure and Definition Language (DASDL))
- OR operator
 - combining with AND and NOT operators, 6-16
 - syntax of, 6-7
 - using in selection expressions, 6-15
- overriding the database title, 4-2

P

- partial binary searches, 6-20
- path adjustment rules, 5-5

- physical database, invoking, 4-1
- prerequisites for restart and recovery, 8-11
- preventing deadlocks, 7-2
- prior physical record, retrieving, 6-6
- process families, use in updating audited databases, 2-6
- processing failures
 - restoring audited databases, 8-4
 - restoring unaudited databases, 2-7
- processing updates, audited versus unaudited databases, 2-5
- PUT, language extension, 3-9

R

- Railroad diagrams, explanation of, D-1
- random searches, 6-7
 - combining with serial searches, 6-17
 - using reblocking, 6-22
 - using selection expressions, 6-7
- RDSSTORE option
 - defining audited databases in DASDL, 8-3
 - overriding the data management AUDIT option, 8-10
- re-creating records, 2-15, (See also recreate operations)
- readaheads, for efficient searches, 6-22
- READONLY
 - exception and error subcategories, A-14
 - major category number, A-2
- REAPPLYCOMPLETED option
 - defining audited databases in DASDL, 8-3
 - effect in halt/load recovery, 8-3
 - effect on the end transaction operation, 8-11
 - use in recovery, 8-5
 - use with INDEPENDENTTRANS, 8-5
- reapplying transactions, 8-5
- reblocking, 6-22
- record area, 1-2
- records
 - adding new records, 7-2
 - adjusting the path for deleted records, 5-5
 - finding, locking, or securing, 2-13
 - freeing, 2-14
 - retrieving, 6-1
- recovering
 - after a halt/load, 8-2
 - after an abort, 8-1
 - audited databases, 2-7
 - from deadlocks, 7-2

- in multitasking, 8-11
- unaudited databases, 2-7
- using audit trails, 2-7, 8-4
- using the DASDL INDEPENDENTTRANS option, 8-5
- using the DASDL REAPPLY-COMPLETED option, 8-5
- recovery techniques
 - audited database, 8-4
 - for unaudited databases, 2-7
- recreate operations
 - changing values of required items, 7-4
 - difference from create operations, 2-15
 - setting the current path, 5-1
- RECREATE, language extension, 3-9
- reentrance, for Access routines, 1-7
- relational operator, using in selection expressions, 6-10
- remaps, in logical databases, 4-1
- remove operations
 - deleting key entries from manual subsets, 7-5
 - setting the current path, 5-1
- REMOVE, language extension, 3-9
- reorganizing databases, 2-3
- required items
 - changing, 7-4
 - definition of, 7-3
- restart code, 8-8
- restart data sets
 - adding and deleting records, 8-7
 - effect of the AUDIT option, 8-10
 - example DASDL, 8-13
 - invoking in an end transaction operation, 8-10
 - invoking in begin transaction operations, 8-9
 - purpose of, 8-1, 8-5
 - use in recovering audited databases, 2-7
 - when to examine, 8-7
 - writing records to, 8-10
- restart procedures, 8-8
- RESTART TRANSACTION ONLY option
 - for audited databases, 8-3
 - writing records to the restart data set, 8-7
- restart user work area, 8-5
- restart, prerequisites for multitasking, 8-11
- restartable program, example of, 8-12

- retrieval operations, (See delete operations, find operations, lock operations, secure operations)
- efficiency considerations, 6-27
- guidelines, 6-27
- result of, 6-2
- setting the current path, 5-1
- types and techniques, 6-2
- retrieving records
 - binary searches, 6-18
 - data management operations, 6-1
 - guidelines, 6-27
 - in a set or subset, 6-18
 - in combined searches, 6-17
 - in databases open for inquiry only, 7-1
 - in databases open for updates, 7-2
 - partial binary searches, 6-20
 - random searches, 6-7
 - serial searches, 6-3
- RPG
 - sample language extension program, C-26
- run-time interface
 - with the interpretive interface, 1-7
 - with the language extensions, 1-4

S

- sample programs
 - ALGOL
 - interpretive interface, B-5
 - language extensions, C-7
 - COBOL74
 - interpretive interface, B-15
 - language extensions, C-16
 - FORTRAN77, interpretive interface, B-25
 - RPG, language extensions, C-26
- savepoint operations, 8-12
- search operations, (See delete operations, find operations, lock operations, secure operations)
- backward search, 6-4
- binary searches, 6-18
- combining serial and random searches, 6-17
- efficiency considerations, 6-27
- forward search, 6-3
- generic syntax, 6-1
- guidelines, 6-27
- partial binary searches, 6-20
- random searches, 6-7
- serial parameters, 6-4

- serial searches, 6-3
- setting the current path, 5-1
- syntax of, 6-1
- types and techniques, 6-2
- using selection expressions, 6-7
- using with set operation, 5-2
- secure operations
 - as a search and retrieval operation, 6-1
 - difference from lock or find operations, 2-13, 7-1
 - for records, 2-13
 - for structures, 2-15
 - setting the current path, 5-1
 - using readaheads, 6-22
 - using reblocking, 6-22
- SECURE, language extension, 3-9
- securing records, 2-13, 7-1
- securing structures, 2-15
- SECURITYERROR
 - exception and error subcategories, A-15
 - major category number, A-2
- selection criteria
 - in combined searches, 6-17
 - in random searches, 6-7
- <selection expression>, 6-7
- selection expressions
 - AND operator, 6-14
 - Boolean operators truth table, 6-13
 - complex syntax, 6-7
 - components of, 6-7
 - duplicate keys, 6-9
 - enhancements for the interpretive interface, 6-10
 - for approximate matches, 6-12
 - for exact matches, 6-11
 - forcing a binary search, 6-20
 - forming, 6-7
 - generic syntax, 6-7
 - in combined searches, 6-17
 - in find key of operations, 6-18
 - in random searches, 6-7
 - key items, 6-8
 - multiple Boolean operators, 6-16
 - NOT operator, 6-13
 - OR operator, 6-15
 - relational operators, 6-10
 - target structure, 6-8
 - using nonkey items, 6-10
- separating data definition from data manipulation, 2-1
- serial parameter, in search operations, 6-1
- serial search parameters, table of, 6-4
- serial searches, 6-3
 - backward, 6-4
 - combining with binary searches, 6-25
 - combining with random searches, 6-17
 - definition of, 6-2
 - early termination, 6-25
 - forward, 6-3
 - retrieving first physical record, 6-5
 - retrieving last physical record, 6-6
 - retrieving next physical record, 6-5
 - retrieving prior physical record, 6-6
 - using reblocking, 6-22
 - using serially parameters, 6-4
- set operations
 - setting the current path, 5-1
 - using with search operations, 5-2
 - with find next or find prior operations, 6-3, 6-4
- set to beginning operations, 5-2, 6-3, (See also set operations)
- set to ending operations, 5-2, 6-4, (See also set operations)
- SET, language extension, 3-9
- sets
 - binary search, 6-18
 - finding records, 6-18
 - partial binary search, 6-20
 - resetting the current path, 5-3
 - valid search types and techniques, 6-27
- setting the current path, 5-1
 - using a combined search, 6-17
 - using a random search, 6-7
 - using a serial search, 6-3
- simple key items, 6-8
- standard entry points
 - find operation enhancements, 6-24
 - table of, 3-4
- statements, (See language extensions)
- storage failures
 - restoring audited databases, 8-1
 - restoring unaudited databases, 2-7
- store operations
 - adding new records, 7-3
 - changing nonrequired items, 7-3
 - changing required items, 7-4
 - setting the current path, 5-1
- STORE, language extension, 3-9
- subcategories for exceptions and errors, tables of, A-2
- subsets
 - automatic
 - deleting key entries, 7-5

- inserting key entries, 7-3, 7-5
- binary search, 6-18
- finding records, 6-18
- manual
 - inserting a key entry, 7-5
 - removing key entries, 7-5
- partial binary search, 6-20
- resetting the current path, 5-3
- valid search types and techniques, 6-27
- SYNC option, in an end transaction operation, 8-11
- syncpoints, as a recovery checkpoint, 8-11
- SYSTEM/DMINTERFACE
 - as the compile-time interface, 1-4
 - use when invoking databases, 1-4
- SYSTEM/DMRECOVERY, (See DMRECOVERY program)
- SYSTEMERROR
 - exception and error subcategories, A-15
 - handling fatal errors, 9-2
 - major category number, A-2

T

- target structure, using in selection
 - expressions, 6-8
- target value, using in selection expressions, 6-11
- tasks for a restart procedure, 8-8
- tasks performed by Accessroutines, 1-7
- terminate operations, 8-12
- testing for relationships in a selection search, 6-10
- TITLE attribute, in database equation, 4-2
- transaction processing
 - audited database, 3-2
 - for recovery, 8-1
 - guidelines, 2-6
 - multitasking, 8-11
 - overview, 2-5
 - unaudited databases, 3-2
- transaction state
 - beginning and ending, 8-9
 - data management operations, 3-2, 8-8
 - multitasking, 8-11
 - operations that must be performed in transaction state, 2-6
 - overview, 2-6
 - result of ABORT task, 8-1
 - syncpoints, 8-11
- transactions, 2-5

- truth table of Boolean operators, 6-13
- two-phase locking, 8-5
- types and techniques, of searches and retrievals, 6-2

U

- unaudited databases
 - difference from audited databases, 2-5
 - recovering, 2-7
 - transaction processing, 2-6
- undefined current record, status of current record, 2-13
- update access mode, 4-4
- user work area, 1-2
 - for restart data sets, 8-5
 - in search and retrieval operations, 6-2

V

- valid current path
 - for data sets, 2-12
 - for sets or subsets, 2-12
- VERSIONERROR
 - exception and error subcategories, A-17
 - major category number, A-2
- virtual records, nonexistent records, 2-12
- VISIBLE DBS, control system resources for databases, 2-3

W

- WFL, (See Work Flow Language (WFL))
- Work Flow Language (WFL)
 - syntax for database equation, 4-2
- writing records to the restart data set, 8-7, 8-10



5044225000000380